

DE GRUYTER

GRADUATE

Jürgen Cleve, Uwe Lämmel

DATA MINING

DATA ANALYSIS FOR ARTIFICIAL INTELLIGENCE



DE
G

DE GRUYTER

GRADUATE

Jürgen Cleve, Uwe Lämmel

DATA MINING

DATA ANALYSIS FOR ARTIFICIAL INTELLIGENCE



DE
G

Jürgen Cleve, Uwe Lämmel
Data Mining

Jürgen Cleve, Uwe Lämmel

Data Mining



Data Analysis for Artificial Intelligence

DE GRUYTER

ISBN 9783112233429

e-ISBN (PDF) 9783112233436

e-ISBN (EPUB) 9783112233443

**Bibliographic information published by the Deutsche
Nationalbibliothek**

The Deutsche Nationalbibliothek lists this publication in the
Deutsche Nationalbibliografie; detailed bibliographic data are
available on the Internet at <http://dnb.dnb.de>.

© 2026 Walter de Gruyter GmbH, Berlin/Boston, Genthiner
Straße 13, 10785 Berlin

Contents

Jürgen Cleve, Uwe Lämmel

Frontmatter

Preface

Contents

1 Introduction

1.1 Evaluation of mass data

1.2 The process of data analysis

The CRISP data mining model

Phase 1: Business Understanding

Phase 2: Data understanding

Phase 3: Data preparation

Phase 4: Modeling (data mining in the narrower sense)

Phase 5: Evaluation

Phase 6: Deployment

1.3 Fayyad's process model

Data selection

Data preprocessing

Data transformation

Data mining

Evaluation and interpretation

1.4 Interdisciplinary of data mining

Databases and data warehouses
Knowledge-based systems
Machine learning
Statistics
Visualization

1.5 Why data mining?

1.6 Tools

1.6.1 Knime
1.6.2 Weka
1.6.3 JAVANNS
1.6.4 Python

2 Fundamentals of data mining

2.1 Basic concepts

2.2 Datatypes

2.3 Distance and similarity measures

2.4 Fundamentals of artificial neural networks

Propagation function
Activation function
Output function
Example neuron

2.5 Logic

2.6 Unsupervised and supervised learning

Unsupervised learning

Supervised learning

3 Application classes

3.1 Cluster analysis

3.2 Classification

3.3 Numerical prediction

Linear regression

3.4 Association analysis

3.5 Text Mining

3.6 Web Mining

4 Knowledge representation

4.1 Decision table

4.2 Decision trees

4.3 Rules

4.4 Association rules

Threshold values

Types of association rules

Hierarchical association rules

Quantitative association rules

Fuzzy association rules

Temporal association rules

4.5 Instance-based representation

4.6 Representation of clusters

4.7 Neural networks as knowledge stores

5 Classification

5.1 The k-nearest neighbor method

5.1.1 The k-nearest neighbor algorithm

5.1.2 A refined algorithm

5.2 Decision tree learning

5.2.1 Creating a decision tree

5.2.2 Selection of an attribute

5.2.3 The ID3 algorithm for generating a decision tree

5.2.4 Entropy

5.2.5 The Gini index

5.2.6 The C4.5 algorithm

5.2.7 Problems with decision tree learning

5.2.8 Decision tree and rules

5.3 Naive Bayes

5.3.1 Bayesian formula

5.3.2 The naive Bayes algorithm

5.4 Feedforward neural networks

5.4.1 Architecture

5.4.2 The backpropagation of error learning method

5.4.3 Modifications to the backpropagation algorithm

5.4.4 An example

5.4.5 Convolutional neural networks

5.5 Support Vector Machines

5.5.1 Basic principle

5.5.2 Formal representation of support vector machines

5.6 Ensemble learning

5.6.1 Bagging

5.6.2 Boosting

5.6.3 Random Forest

6 Cluster analysis

6.1 Types of cluster analysis

6.1.1 Partitioning clustering

6.1.2 Hierarchical clustering

6.1.3 Density-based clustering

6.1.4 Cluster analysis with neural networks

6.2 The k-means algorithm

6.3 The k-medoid algorithm

6.4 Expectation maximization

6.5 Agglomerative clustering

Single linkage (nearest neighbor)

Complete linkage (furthest neighbor)

Average linkage (within groups)

Centroid

Ward

6.6 Density-based clustering

6.7 Cluster formation using self-organizing maps

6.7.1 Structure

6.7.2 Learning

6.7.3 Visualization of a SOM

6.7.4 An example

6.8 Clustering using neural gas

6.9 Cluster formation using ART

6.10 The fuzzy c-means algorithm

7 Association analysis

7.1 The Apriori algorithm

7.1.1 Generating the candidates

7.1.2 Generating the rules

7.2 Frequent Pattern Growth

7.3 Association rules for special tasks

7.3.1 Hierarchical association rules

7.3.2 Quantitative association rules

7.3.3 Generation of temporal association rules

8 Data preparation

8.1 Motivation

8.2 Types of data preparation

8.2.1 Data selection and data integration

- 8.2.2 Data cleaning
- 8.2.3 Data reduction
- 8.2.4 Uneven distribution of the target attribute
- 8.2.5 Data transformation

8.3 An example

9 Evaluation

9.1 Principle of minimum description length

9.2 Measures for the interestingness of association rules

- 9.2.1 Support
- 9.2.2 Confidence
- 9.2.3 Completeness
- 9.2.4 Gain function
- 9.2.5 p -s-Funktion
- 9.2.6 Lift
- 9.2.7 Categorization of interestingness measures

9.3 Quality measures and error costs

- 9.3.1 Error rate
- 9.3.2 Additional quality measures of classifiers
- 9.3.3 Error costs

9.4 Test sets

- Holdout
- Stratification
- Cross-validation
- Leave-one-out

9.5 Quality of clusters

9.6 Visualization

10 A data mining task

10.1 The task

10.2 The problem

10.3 The data

10.4 Data preparation

10.5 Experiments

10.5.1 K-nearest neighbor

10.5.2 Naive Bayes

10.5.3 Decision tree method

10.5.4 Neural networks

10.6 Python program

10.6.1 Data preprocessing

10.6.2 Model development

10.6.3 Application of the model for a prediction

10.7 Evaluation of the results

A Appendix: Sample data

A.1 Iris data

A.2 Soybeans

A.3 Weather data

A.4 Contact lens data

Bibliography

Index Rerum

Frontmatter

Jürgen Cleve

Uwe Lämmel

Preface

Artificial intelligence (AI) has been on everyone's lips for quite some time now. Generative AI in particular is constantly opening up new areas of application. Generative AI is based on machine learning processes, which brings us to data mining: *data mining* encompasses a range of basic technologies for analyzing mass data, which in turn are relevant for training AI. We hear or read buzzwords such as *big data*, *predictive analytics*, *data science*, or *machine learning* almost every day.

Data is collected in many areas, such as research, to discover new correlations or improve existing processes, while companies collect data to achieve higher sales figures through individualized marketing or to reduce machine or plant downtime. Even in medicine, data is collected and evaluated to enable better diagnoses and therapies.

With the help of such large data collections, methods for text or character recognition, speech comprehension, and image recognition have been developed that are already integrated into many systems today and are giving rise to intelligent applications.

Data is collected for various reasons: first of all, simply because collecting and storing data is technically and financially possible. Secondly, there are legal requirements that data must be stored, at least for a certain period of time. Thirdly, and here we are getting closer to data mining, collecting data is also carried out for a specific purpose, i.e. a benefit is expected.

What data is collected? All data, unless collection is restricted by law: technical data, business data, and even private data.

In this book, we examine techniques that can be used to evaluate large amounts of data. We focus on the processes and techniques for analyzing structured data that are commonly associated with data mining. Specifically, our book covers the following topics:

- First, we provide an overview of data mining and discuss basic concepts and procedures.
- Next, we describe application classes in order to identify the possible uses of data mining.
- Models are derived from the data. We show how these models can be represented or stored.
- The methods for analyzing the data, data mining in the narrower sense, naturally take up most of the space in this book.
- The data must be prepared to enable analysis and improve data quality. Data quality can have a significant impact on results. A separate chapter is therefore devoted to data preparation.
- This is followed by an evaluation of the analysis results: Are they new, are they really relevant?
This chapter also considers possible visualizations of the results as a form of evaluation. A suitable graphical

representation helps both to assess the quality of the results and to build confidence in a model.

- Finally, we run through data mining using a concrete example. We thus illustrate the data mining process and apply the analysis techniques presented in the book.

Statistical methods for data evaluation are only touched upon briefly. The areas of text analysis, known as text mining, as well as web mining, are only outlined.

Unlike many other books, we separate the models, such as decision trees or artificial neural networks, from the data mining methods that use them, such as the generation of a decision tree for classification. This makes it clearer that some models can be used for different tasks.

→ Chapter 3 introduces application classes. It answers the question of what typical areas of application there are for data mining? Then, in → Chapter 4, we discuss data mining models and the possibilities for representing them. Typical questions are: How can a classification model be represented? How can a cluster formation be represented? → Chapters 5 to → 7 cover the methods assigned to the respective application classes, while → Chapter 8 is dedicated to data preparation. Although this phase is the first step in a data mining process and often crucial to its success, we have placed it further back in the book. Once you are familiar with the data mining algorithms, it is easier to understand why and how certain data must be preprocessed. In → Chapter 9, we look at some techniques for evaluating the results obtained through data mining.

This book is a textbook. Data mining is now included in almost all computer science-related degree programs. The aim of this book is to provide an introduction to the interesting field of data

mining. We have deliberately refrained from presenting the underlying mathematical details for some of the methods. Likewise, we have concentrated on elementary algorithms.

Data mining is a field that requires experience. A project in which data mining tools are used blindly will rarely be successful. An understanding of the subject area in question, the available data, and also the data mining methods is a necessary prerequisite for a successful data analysis project.

This book provides an introduction to the field of data mining, allowing you to carry out data analysis projects in a structured and targeted manner. It will enable you to evaluate data mining methods in terms of their areas of application and performance, and to prepare the data for these methods accordingly.

Most of the examples are implemented in KNIME and WEKA. JAVANNS and Python were used to develop artificial neural networks.

We recommend following the examples using these tools. Install these systems, try out small examples, and familiarize yourself with how to use them.

Anyone interested in information on the topic of data mining will find many good web-pages on the Internet. There are now a number of competitions in which real-world problems have to be solved. The authors and their students have participated several times in the Data Mining Cup¹, which has been organized for several years by the company *prudsys AG*² in Chemnitz, Germany. There are also many platforms where real problems can be solved, such as the platform → www.kaggle.com. We have listed additional platforms on the book's website.

You can find not only information about the book, but also examples at → <https://www.fww.hs-wismar.de/datamining/>.



We would like to take this opportunity to thank *prudsys* and the developers of *KNIME* and *WEKA* for allowing us to use examples from their environments. Please note that not all screenshots of the systems are from the current versions.

Working with De Gruyter Brill was pleasant and straightforward.
We would like to express our special thanks to Mrs. Skambrak,
Mrs. Stanciu, Mrs. Lippert, and Mr. Sacco.

Wismar and Rostock, May 2026

Jürgen Cleve and Uwe Lämmel

Contents

1 Introduction

- 1.1 Evaluation of mass data
- 1.2 The process of data analysis
- 1.3 Fayyad's process model
- 1.4 Interdisciplinarity of data mining
- 1.5 Why data mining?
- 1.6 Tools
 - 1.6.1 KNIME
 - 1.6.2 WEKA
 - 1.6.3 JAVANNS
 - 1.6.4 Python

2 Fundamentals of data mining

- 2.1 Basic concepts
- 2.2 Data types
- 2.3 Distance and similarity measures
- 2.4 Fundamentals of artificial neural networks
- 2.5 Logic
- 2.6 Unsupervised and supervised learning

3 Application classes

- 3.1 Cluster analysis
- 3.2 Classification
- 3.3 Numerical prediction
- 3.4 Association analysis
- 3.5 Text Mining
- 3.6 Web Mining

4 Knowledge representation

- 4.1 Decision table
- 4.2 Decision trees
- 4.3 Rules
- 4.4 Association rules
- 4.5 Instance-based representation
- 4.6 Representation of clusters
- 4.7 Neural networks as knowledge stores

5 Classification

- 5.1 The k-nearest neighbor method
 - 5.1.1 The k-nearest neighbor algorithm
 - 5.1.2 A refined algorithm
- 5.2 Decision tree learning
 - 5.2.1 Creating a decision tree
 - 5.2.2 Selection of an attribute
 - 5.2.3 The ID3 algorithm for generating a decision tree
 - 5.2.4 Entropy
 - 5.2.5 The Gini index
 - 5.2.6 The C4.5 algorithm
 - 5.2.7 Problems with decision tree learning
 - 5.2.8 Decision tree and rules
- 5.3 Naive Bayes
 - 5.3.1 Bayesian formula
 - 5.3.2 The naive Bayes algorithm
- 5.4 Feedforward neural networks
 - 5.4.1 Architecture
 - 5.4.2 The backpropagation of error learning method
 - 5.4.3 Modifications to the backpropagation algorithm
 - 5.4.4 An example
 - 5.4.5 Convolutional neural networks

5.5 Support Vector Machines

5.5.1 Basic principle

5.5.2 Formal representation of support vectormachines

5.6 Ensemble learning

5.6.1 Bagging

5.6.2 Boosting

5.6.3 Random Forest

6 Cluster analysis

6.1 Types of cluster analysis

6.1.1 Partitioning clustering

6.1.2 Hierarchical clustering

6.1.3 Density-based clustering

6.1.4 Cluster analysis with neural networks

6.2 The k-means algorithm

6.3 The k-medoid algorithm

6.4 Expectation maximization

6.5 Agglomerative clustering

6.6 Density-based clustering

6.7 Cluster formation using self-organizing maps

6.7.1 Structure

6.7.2 Learning

6.7.3 Visualization of a SOM

6.7.4 An example

6.8 Clustering using neural gas

6.9 Cluster formation using ART

6.10 The fuzzy c-means algorithm

7 Association analysis

7.1 The Apriori algorithm

7.1.1 Generating the candidates

7.1.2 Generating the rules

7.2 Frequent Pattern Growth

7.3 Association rules for special tasks

7.3.1 Hierarchical association rules

7.3.2 Quantitative association rules

7.3.3 Generation of temporal association rules

8 Data preparation

8.1 Motivation

8.2 Types of data preparation

8.2.1 Data selection and data integration

8.2.2 Data cleaning

8.2.3 Data reduction

8.2.4 Uneven distribution of the target attribute

8.2.5 Data transformation

8.3 An example

9 Evaluation

9.1 Principle of minimum description length

9.2 Measures for the interestingness of association rules

9.2.1 Support

9.2.2 Confidence

9.2.3 Completeness

9.2.4 Gain function

9.2.5 p -s-Funktion

9.2.6 Lift

9.2.7 Categorization of interestingness measures

9.3 Quality measures and error costs

9.3.1 Error rate

9.3.2 Additional quality measures of classifiers

9.3.3 Error costs

9.4 Test sets

9.5 Quality of clusters

9.6 Visualization

10 A data mining task

10.1 The task

10.2 The problem

10.3 The data

10.4 Data preparation

10.5 Experiments

10.5.1 K-nearest neighbor

10.5.2 Naive Bayes

10.5.3 Decision tree method

10.5.4 Neural networks

10.6 Python program

10.6.1 Data preprocessing

10.6.2 Model development

10.6.3 Application of the model for a prediction

10.7 Evaluation of the results

A Appendix: Sample data

A.1 Iris data

A.2 Soybeans

A.3 Weather data

A.4 Contact lens data

Bibliography

Index

1 Introduction

Data you don't need is never lost.

Alder's first negative Principle of Computers

1.1 Evaluation of mass data

The amount of available data is not only growing steadily, but also at an ever-increasing rate. Every company and every institution collects data, either voluntarily or due to legal requirements. Banks store transaction data, companies store customer data, for example on their purchasing behavior, weather institutes collect weather data, technology companies collect data to improve their algorithms for character, text, or speech recognition and now also for generating speech, text, images, or music. In the course of digitalization, the collection of data is being driven forward, whether it be user or process data.

Do you know who stores what data about you? If you are employed, what data does your company collect?

Powerful computers enable us not only to store this data, but also to analyze it and generate interesting and, hopefully useful information from it.

Whereas the emphasis used to be on simply archiving data or information, in recent years the focus has shifted significantly toward active use of data. This raises two questions:

- HOW do we evaluate data?
- WHAT can we get out of the collected data?

Extensive data can no longer be evaluated just by “looking at it” or using simple Excel spreadsheets. Even statistics often reach its limits. Powerful *data analysis techniques* are needed for large

amounts of data. These techniques search the data for patterns, i.e., correlations, in order to predict specific customer behavior, for example.

This search for patterns or correlations in data is the subject of *data mining*. While mining involves searching for something, such as coal in coal mining, data mining does not search for data, but for the treasures or information hidden within the data.

Data mining aims to detect unknown patterns or dependencies in the given data. One of the goals is to classify objects into categories that may be known or unknown beforehand. Some application examples (see also → Section 1.5) may illustrate the practical relevance:

- Credit or no credit?
Rules are derived from old credit data to assist in assessing the *credit rating* of a bank customer.
- Discount or no discount?
Predictions are generated from customer data to determine which customers are most likely to be encouraged by discounts to make further purchases.
- Adventure or all-inclusive?
Patterns of typical *travelers* are generated in order to put together offers tailored to the respective customer types.
- Diapers and beer or bread?
Customers' purchasing *behavior* is analyzed to identify dependencies between purchased products. People who buy diapers often also buy beer.
- Which gene?
The genes of *diabetes patients* are examined with the aim of identifying the genes that are presumably responsible or partly responsible for the disease.

The aim of data mining is to extract useful knowledge from mass data, such as customer or company data, company key figures, or

process data. If this is successful, it can result in a competitive advantage for the company in the market. Data mining can thus be classified in the areas of *knowledge management* (knowledge is extracted from data) and *business intelligence* (data from different areas is analyzed).

Data mining is based on the following *fundamental assumptions*:

1. *Sufficient data* is available. On the one hand, there must be a sufficient amount of data available, and on the other hand, it must be representative of the section of the world being examined.
2. The *future* behaves like the past. The available data from the past can therefore be used to build a model for the future.
3. The data actually *contains information* that can be extracted.
4. The data *is usable* and does not subject to any legal restrictions.

1.2 The process of data analysis

Data mining is usually embedded in analytical information systems such as a data warehouse or a business intelligence system, but can also be operated independently.

A data mining process is divided into several phases:

Selection Selection of suitable data sets

First, the available data is reviewed and transferred to a database, usually even to a *single* data table.

Data preprocessing Handling missing or problematic data

In this phase, the data is cleaned up by replacing missing values and correcting contradictory values.

Transformation Conversion into appropriate data formats

The methods used often require very specific data formats, meaning that the data may need to be transformed. For

example, metric values are grouped into intervals.

Data mining Searching for patterns

This is where the actual data mining takes place, i.e., the development of a *model* such as the creation of a decision tree.

Interpretation and evaluation Interpretation of the results and evaluation

In the final phase, the results found are examined. Are they new? Are they helpful?

→ Figure 1.1 shows the phases and how they interact in the data mining process. Often, both the entire process and the actual analysis step are referred to as *data mining*.

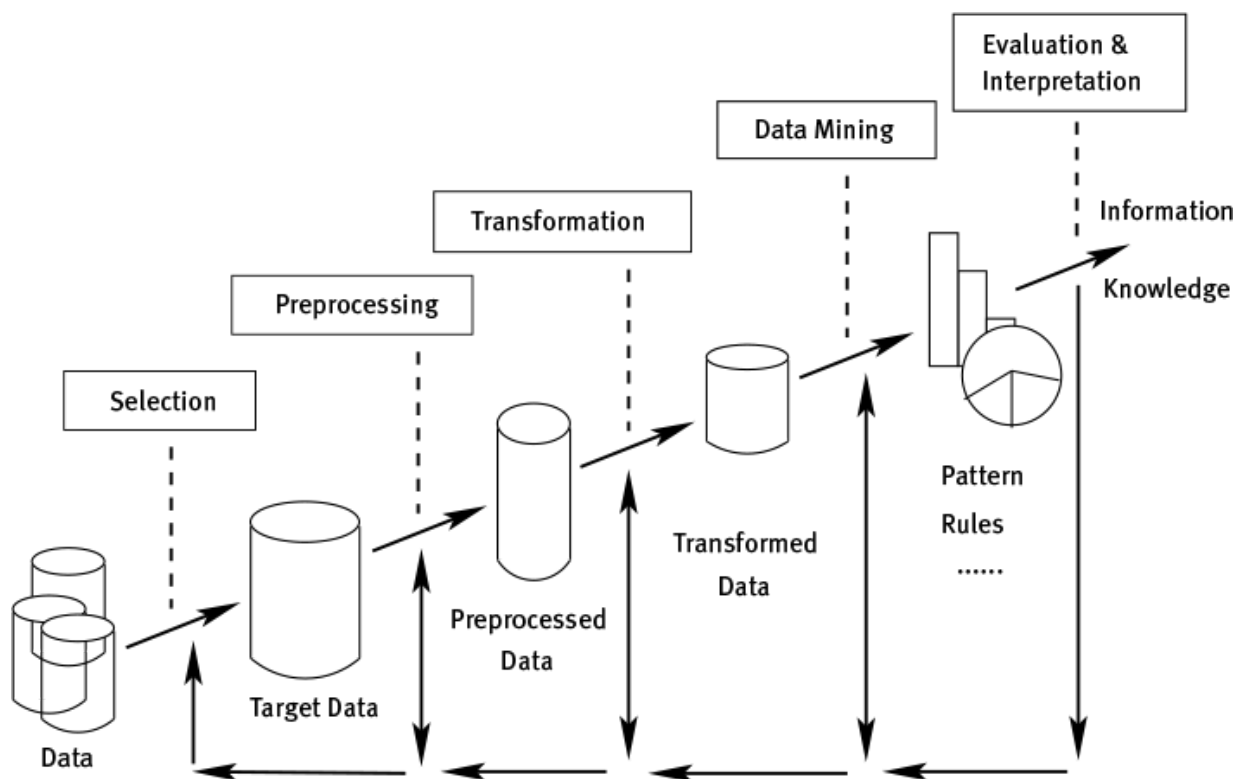


Fig. 1.1 Outline of a data mining process according to Fayyad [→ FPSS96]

Knowledge discovery in databases (KDD) is often understood to mean the entire process. The actual *data mining* is then a sub-step. However, the two terms *data mining* and *KDD* are often used synonymously. We will also not distinguish between the terms *data mining* and *KDD* and use them synonymously, but we prefer the term *data mining*.

The CRISP data mining model

A second model for data mining processes is the CRISP-DM model, which was developed by a consortium consisting of the following companies:

- NCR Corporation,
- Daimler AG,
- SPSS,
- Teradata, and
- OHRA.

CRISP-DM stands for **C**ross **I**ndustry **S**tandard **P**rocess for **D**ata **M**ining. CRISP-DM is a phase-oriented process model for data mining projects. The aim is to define a data mining process in which an analysis model is developed and validated. In contrast to the previous considerations of the phases, the CRISP-DM diagram in → Figure 1.2 illustrates the cyclical nature of a data mining project.

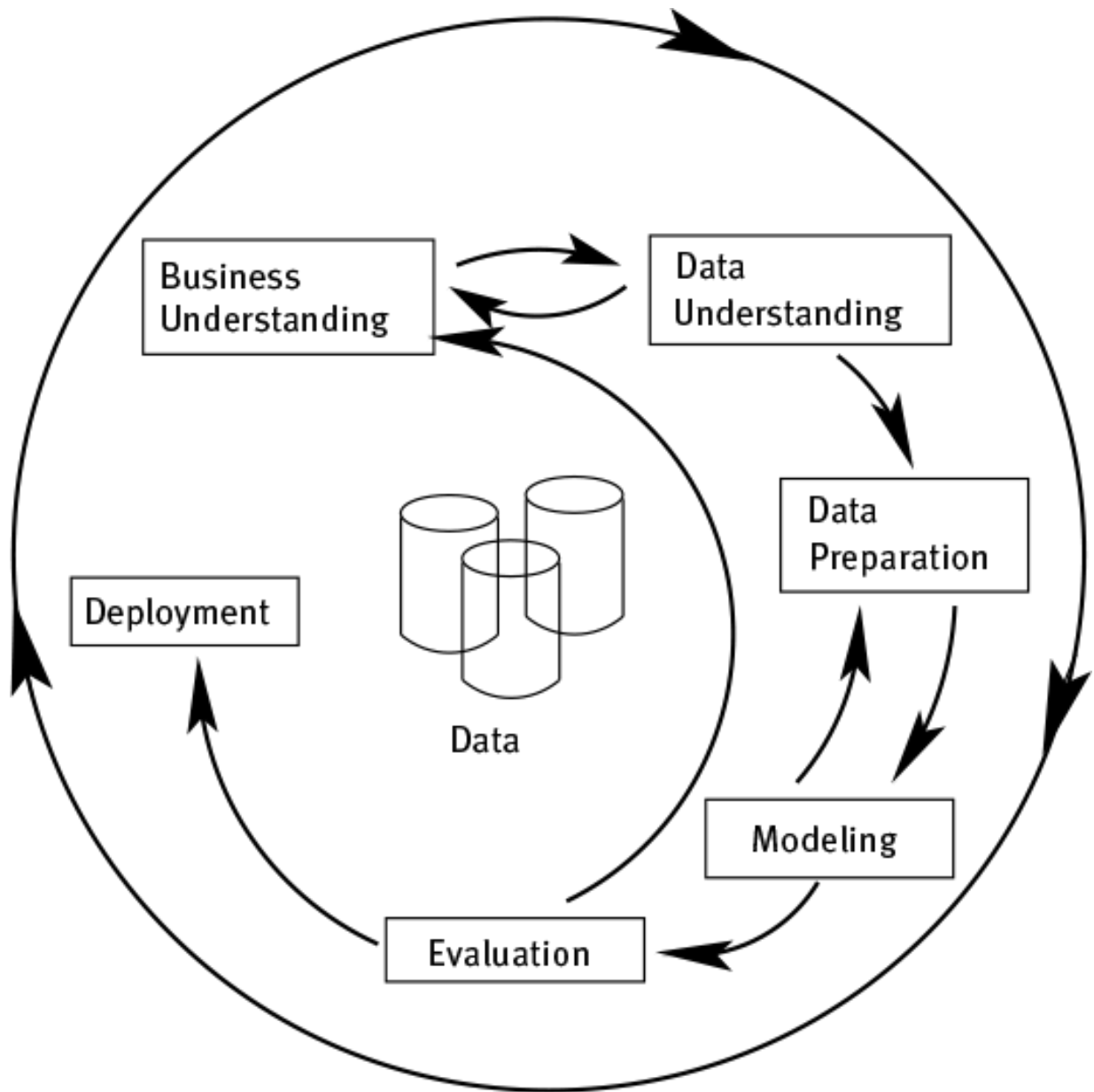


Fig. 1.2 CRISP data mining process

The CRISP-DM model organizes six phases into several cycles. The six phases are outlined below:

Phase 1: Business Understanding

Without a fundamental understanding of the subject area in which data analysis is to take place, it is rarely possible to achieve a good

result. In this initial phase, the problem must be understood in terms of content and the task to be solved must be derived from this. The objectives are defined:

- Based on the current situation, the results to be achieved in the data mining project are identified.
- What results are expected?
- What is the initial situation?
- What business objectives need to be considered?
- What resources are available?
- Success criteria must be defined, such as a target recognition rate for classification tasks.
- Risks are assessed and quantified as far as possible. For example, obtaining suitable data may prove to be more time-consuming and therefore more expensive than planned.
- It may also be the case that certain data is not available for the project for legal or company policy reasons.
- The costs must be planned and weighed against the expected benefits.

Any data analysis that is not yet integrated into operational processes always has a project-like character. Appropriate project management is required, especially since data analysts, usually computer scientists, work together with users from other specialist areas, such as the specialist departments of companies.

This phase concludes with a *project plan*.

Phase 2: Data understanding

The second phase deals with the available data. Depending on the objective of the analysis, it is defined which data is required. At the same time, it is examined which data is available. For a successful project, it is necessary to

understand the data and its significance in the respective context precisely.

A data mining project in which the available data is simply used without knowing its semantics will not be successful.

Thus, the data must not only be collected, but also described. In addition, the data is examined for quality. Statistical analyses, such as the determination of measures of descriptive statistics, provide information about the nature of the data and give an initial impression of the analysis options. These measures include minimum, maximum, mean, median, and correlation coefficients.

In this phase, it is useful to employ visualization techniques to represent the values of the characteristics, for example, histograms and bar charts (see → Section 9.6).

Phase 3: Data preparation

Once you have gained an understanding of the task, the operational background, and the data, you can now prepare for the actual data mining step. First, the data relevant to the task is selected and transferred to a consistent data table. The data is cleaned up, and erroneous or inconsistent data is corrected. A decision must be made on how to handle missing values. If necessary, new attributes are introduced or attributes are merged. The data is transformed appropriately so that it can be processed by the data mining procedures.

Data preparation can be a decisive factor in the success or failure of a data analysis. For this reason, we discuss data preparation separately in → Chapter 8.

Phase 4: Modeling (data mining in the narrower sense)

This phase involves the actual data analysis: a model is created that describes the relationship between input data and results, such as a classification. Models can be, for example

- Decision trees,
- Various forms of rules,

- Trained artificial neural networks, or
- Descriptions of clusters.

Depending on the task at hand, an appropriate method is selected. The experiments are designed and configured, and parameters for the method are set. The experiments are carried out and, if necessary, repeated with modified parameters in order to refine and hopefully improve the model.

Data preparation and model building are closely linked in a cyclical relationship, as a large number of experiments with differently prepared data must be carried out in order to achieve a promising result.

Phase 5: Evaluation

The models and results developed must be reviewed and evaluated. The results are measured against the success criteria defined in the *business understanding* phase. For example, do the results achieve the expected economic benefits? An error analysis can reveal opportunities for new experiments, and the process returns to the *data preparation* phase or directly to the *modeling phase*. The project plan is also modified and adapted as necessary.

Phase 6: Deployment

In the final phase, the model that has been developed must be transferred to everyday use. This is a critical phase for many data mining projects. Without well-planned monitoring and sufficient motivation and support, successful implementation can fail.

Usually, a data mining project is only *one phase* of an overall process. The model must then be transferred to regular operation and integrated into the ongoing processes. Of course, the usual requirements for project completion also apply to data mining projects.

The CRISP-DM model differs from the process described by Fayyad in → Figure 1.1. The phases

- Business understanding,
- Data understanding, and
- Deployment

are not listed in the Fayyad model. In contrast, the data preparation phase of the CRISP-DM model is divided into three more detailed phases in the Fayyad model. The focus of the Fayyad model is more on the technical approach, the actual data provision, and data analysis, while the CRISP-DM model more closely reflects the user's perspective on data mining projects.

We will use the Fayyad model shown in → Figure 1.1 as a guide, as we are providing an introduction to data mining here and are therefore focusing more on the technical approach. The *business understanding*, *data understanding*, and *deployment* phases in the CRISP-DM model are highly dependent on the specific application.

There are other models for the data mining project process. For example, the SEMMA process model is used by SAS Institute Inc. in connection with its Enterprise Miner product. SEMMA stands for *Sample, Explore, Modify, Model, and Assess*. A SEMMA process consists of the following steps:

1. **Sampling:** The data relevant for the analysis is collected.
2. **Exploring:** The data is examined before the actual data mining model is created. The goal is to check the data quality and gain an understanding of the data. Initial visualizations are also carried out.
3. **Modifying:** The data is modified to improve data quality. It is transformed into a form that is suitable for the selected method.
4. **Modeling:** Now the actual analysis and model building takes place.
5. **Assessment:** The results are evaluated.

1.3 Fayyad's process model

In this section, we will go into more detail about the aforementioned Fayyad process model, which serves as a guide for our book.

Data selection

In the first phase of the KDD process, the data required for the analysis requested by the user or deemed suitable for analysis must be determined and exported from the given data sources. In addition to the basic data set, external data can also be used for the analysis. For example, address brokers offer information that can be used to better identify potential customers or interested parties. In the data selection phase, it is checked which data is

- necessary and
- available

to achieve the set goal.

If the selected data cannot be transferred to a target database due to technical or legal restrictions, the data selection must be reconsidered and repeated. Technical restrictions that prevent transfer to a target database include, for example, limitations on capacity and data type in the target system or missing user access rights. One way to circumvent these problems, at least in part, is to limit the selection to a representative subset of the total data set.

Data preprocessing

The raw data sets available in companies often prove unsuitable for data mining analyses in their original form.

Since the target data is merely extracted from the data sources, the quality of the target data set must be examined during data preprocessing and, if necessary, improved using appropriate methods. Due to technical or human errors, the data in operational

systems may contain *incorrect elements*. It is estimated that up to 5 % of the fields in a real data set contain incorrect information.

The aim of data preprocessing is, in particular, to ensure invariant forms of data representation (for example, by converting textual information into unique keys or codes) and to restrict value ranges in order to reduce the number of characteristics to be considered (dimension reduction). The latter can be achieved by generalizing attribute values to a higher level of aggregation, for example by using taxonomies or by forming value intervals, which changes the granularity of the data. Knowledge of the weaknesses of the analysis data is important for the quality of the investigation results. Users of the analysis tools must be able to trust the reliability and accuracy of the data. Faulty data may distort the results without the user being aware of these shortcomings.

Missing data may prevent the calculation of key figures such as a company's turnover. The increasing use of (partially) automated data analysis has resulted in greater susceptibility to data deficiencies, which must be addressed by appropriate mechanisms for detecting and eliminating such weaknesses. A common, easily identifiable type of error is *missing values*. Various techniques are available for dealing with missing values, which are discussed in → Section 8.2.2.

Another potential type of error is caused by *outliers*. These are values that deviate significantly from the rest. Such values may be correctly recorded data that should be included in the analysis, or they may be incorrect data that must be disregarded and deleted from the data set. The insights that users of a data mining system gains about the data set during this phase can provide clues for improving the data quality of the operational systems.

Data transformation

In the data transformation phase, the target data relevant for analysis is transformed into a database schema that can be processed by the data mining system being used.

During data preparation, new attributes or data records can be generated, or existing attributes can be transformed. This step is necessary because analysis methods place specific requirements on the data structure of the input data.

Data transformation changes the data in such a way that a data mining procedure can process it. Certain procedures expect numerical attributes, while others require nominal data.

Data mining

If suitable data sets of acceptable quality are available, the analyses can be carried out. In this phase, the procedure is selected and used to identify patterns in the prepared data set. The first step is to decide which basic data mining task (e.g., classification or cluster formation) is required.

This is followed by the selection of a suitable data mining method. Once a method suitable for the specific problem has been selected, it is configured. This parameterization refers to the specification of certain method-specific values, such as the definition of minimum relative frequencies for an interest filter, the selection of attributes to be considered in pattern formation or description, or the setting of weighting factors for individual input variables. Having found a satisfactory configuration, the search for interesting patterns in the data can begin. The analysis methods generate a model, which then serves as the basis for evaluating this or other data.

Evaluation and interpretation

In this phase of the KDD process, the patterns and relationships that have been discovered are evaluated and interpreted. These patterns must meet the requirements of *validity*, *novelty*, *usefulness*, and *comprehensibility* in order to represent new knowledge and be accessible for interpretation. The latter is a prerequisite for implementing the insights gained in the context of concrete actions.

By no means do all of the patterns uncovered meet these criteria. The analysis methods often reveal many regularities that are irrelevant, trivial, meaningless, or already known, and therefore of no use for the company, or that are not comprehensible. Patterns can be evaluated based on the criterion of interestingness. The following dimensions of *interestingness* are useful:

- The *validity* of a pattern is a measure of the certainty with which the model found (e.g., a pattern or association rule) is also valid in relation to new data.
- The criterion of *novelty* assesses the extent to which a pattern supplements or contradicts existing knowledge.
- The *usefulness* of a pattern assesses its practical benefit for the user.
- *Comprehensibility* measures how well a statement can be understood by a user.

The correct interpretation of data mining results requires a high level of domain knowledge. The interpretation serves to effectively change the user's domain knowledge. Ideally, a team of experts from different areas is formed to ensure that the evaluation is correct and that the information obtained can be used in the best possible way. The interpretation phase can be supported by suitable presentation tools and the availability of additional information about the application domain. Typically, this phase involves a return to one of the previous phases. This usually requires adjusting the parameters or choosing a different data mining technique. It may also be necessary to return to the data selection phase if it is determined that the desired results cannot be achieved with the database used.

1.4 Interdisciplinary of data mining

Data mining is part of many areas that deal in some way with the management and processing of data and knowledge.

Data mining is often mixed with other terms such as

- *Artificial intelligence*,
- *Bigdata*,
- *Data science*,
- *Predictive analytics*,
- *Deep learning*, or
- *Machine learning*.

We will not define these terms precisely, but rather outline the differences between them and their relationship to *data mining*.

Intelligent behavior is inconceivable in many areas without good data analysis. In this respect, classifying data mining as a part of the field of *artificial intelligence* [→ LC23] is entirely reasonable. In particular, there is considerable overlap with *machine learning*, a subfield of artificial intelligence.

Big data emerged against the backdrop of the digitization of the economy and society, which has led to a sharp increase in

- the *volume* of data,
- the speed (*velocity*) at which data is generated, processed, and stored, and
- the *variety* of data types and sources.

Big data therefore focuses on the handling of mass data. This includes data analysis techniques. The term *big data* has now been replaced by the term *data science*, see [→ Sha26].

The term *analytics* is associated with many applications related to data mining. The focus here is on different questions:

- **Descriptive analytics:** What happened?
- **Diagnostic analytics:** Why did it happen?
- **Real-time analytics:** What is happening right now?
- **Predictive analytics:** What could/will happen?

- **Prescriptive Analytics:** What should happen?

Predictive analytics in particular often appears in the context of *data mining*. It considers methods for making predictions, which requires techniques from statistics and data mining. Specific terms have already been coined for the application-oriented use of predictions, such as *predictive policing* or *predictive maintenance*.

Machine learning and data mining have a lot in common. While data mining is used to create static models, machine learning also considers algorithms that learn: processes in which the models developed continue to adapt during use, for example through reinforcement learning. *Deep learning* has become synonymous with neural networks that have many hidden layers and therefore require training (*learning*) across these many layers (i.e., *deep*). Deep learning has led to remarkable successes in many areas of pattern recognition (images, characters, speech) in connection with convolutional neural networks (CNN), which were developed for object recognition, see [→ CW25]. Deep learning is part of machine learning.

The term *business intelligence* has become an umbrella term for areas related to data storage and use. One possible hierarchy of terms for classifying *business intelligence* and *data mining* is as follows:

Information, data, and knowledge management



Business intelligence



Analytical information systems



Data mining

Data mining can be defined as a collection of techniques, methods, and algorithms for analyzing data, which thus represent basic techniques for newer and more complex approaches such as *business intelligence* or *big data*.

The starting point for *business intelligence* (BI) is the observation that, in times of globalization and the Internet, effective and efficient use of the knowledge available in a company is not only a competitive advantage, but also essential for survival. Today, business intelligence encompasses techniques and architectures for the efficient management of corporate knowledge, including, of course, various evaluation options. The tasks of business intelligence are therefore:

- Knowledge acquisition,
- Knowledge management, and
- Knowledge processing.

From an IT perspective, *business intelligence* has many cross-references to information and knowledge management, databases and data warehouses, artificial intelligence, and, of course, data mining (including online analytical processing or statistics).

To date, there is still no generally accepted definition of the term *business intelligence*. *Business intelligence* in the narrower sense refers to core applications that directly support decision-making. These include, for example, online analytical processing (OLAP), management information systems (MIS), and executive information systems (EIS). A somewhat broader definition of BI focuses on analysis. According to this definition, BI includes all applications in which the user performs or prepares analyses. In addition to the applications mentioned above, this now also includes data mining, reporting, and analytical customer relationship management. Finally, business intelligence in the broadest sense encompasses all applications used in the decision-making process, including presentation systems as well as data storage and management. This book focuses on techniques for extracting knowledge using data mining. We therefore consider a small section of the BI spectrum.

The connection between data mining and *data warehouses* is obvious. Data warehouses are designed to provide integrated data to support management decisions and have the following characteristics (see [→ Pet05, p. 40]):

- A data warehouse should be oriented toward the user, the decision-maker, or manager, and thus meet the information needs of management in particular.
- It contains all data relevant to decision-making in a consistent form.
- A data warehouse is only a “collection point” for data from external sources. The data is normally only updated at fixed intervals. Access to the data in the data warehouse is then read-only.
- The data must be managed in a time-dependent manner so that trends can be identified.
- The data is not transferred 1:1 from the sources, but is already accumulated or filtered. Redundancy is therefore possible in a data warehouse.

In order to meet these requirements, tools for data analysis are of course necessary in addition to components for efficient and consistent data storage and fast, flexible access to the data. That is why many data warehouse systems have components for data analysis or at least interfaces to external tools.

Ask an AI chatbot such as ChatGPT or Copilot about the context and differences between the various terms mentioned here: *data warehouse*, *business intelligence*, *data mining*, and *data science*. You will receive quite reasonable answers.

Data mining makes use of proven techniques from many areas of research and adds new approaches to them. Data mining is an interdisciplinary field. Ultimately, all data mining analysis methods are based on mathematics. Statistics in particular contributes a number of its own approaches to data analysis, but is also used for data preprocessing. Statistics is also the basis for some methods, such as the *naive Bayes* classification approach.

Only database technology enables the management of large amounts of data, which is made explicitly clear by the synonymous term *Knowledge Discovery in Databases (KDD)* for data mining.

Artificial intelligence, as the science of knowledge processing, provides techniques for presenting analysis results, i.e., for representing knowledge as logical formulas or, in particular, as rules. Another form of presenting results as a basis for use or evaluation is graphical representation, or visualization. Computer graphics is thus another discipline that is closely related to data mining. → Figure 1.3 illustrates this inherent interdisciplinarity of data mining.

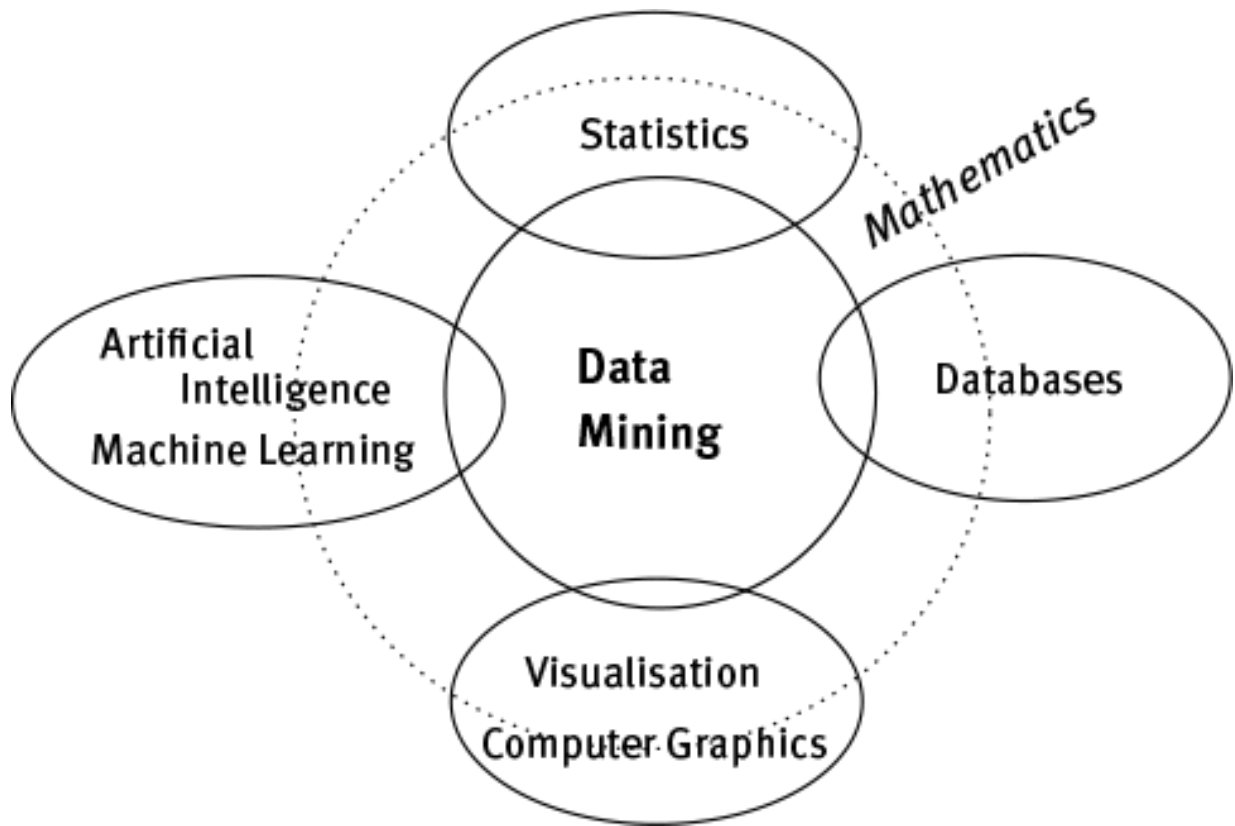


Fig. 1.3 Interdisciplinarity

Databases and data warehouses

In many cases, *databases* form the basis of data mining. Existing data sets are often searched for new correlations or anomalies.

A *data warehouse* usually consists of several databases and also contains the data to be analyzed. The characteristics of a data warehouse include topic orientation, standardization, time orientation, and consistency. From a technical perspective, standardization is particularly noteworthy: data from different sources and with potentially different scales or units of measurement are correctly merged. In addition, all data is stored with a time stamp, creating time series that can be used for evaluation. Consistency means that a data warehouse grows continuously, with time-stamped data being accumulated. In

addition to databases or a data warehouse, text files or web pages can also form the basis of a data mining process.

Knowledge-based systems

Knowledge-based systems attempt to simulate one or more qualified human experts in solving problems within a limited area of application. They contain large amounts of knowledge about a narrowly defined specialist area. They also take into account rules of thumb that are designed to make use of experience from sub-areas for specific problems. If knowledge can be successfully extracted from the data in a data mining process, this knowledge can then be represented and applied in an expert system, for example in the form of rules.

Machine learning

The term *learning* encompasses many complex aspects. Not all of these can be modeled on a computer. *Machine learning* attempts to make computer-based learning methods available so that the program can generate *knowledge* from input information. In machine learning systems, as in human psychology, the simplest learning strategy is memorization. This involves storing the presented knowledge in a list or database. An equally simple form of machine learning is the direct programming of knowledge into the source code of a corresponding program. However, this is not what is meant by *machine learning* in artificial intelligence. Here, the aim is rather to understand contexts and backgrounds in order to be able to recognize patterns or dependencies, for example. Inductive learning involves, among other things, attempting to generalize from examples and thus generate new knowledge.

Statistics

Data mining would be inconceivable without *statistics*, whether in the form of statistical measures that help to understand the data or statistical methods for uncovering correlations.

It is not always possible or sensible to develop and apply a machine-based data mining procedure. Sometimes statistical solutions are sufficient, for example, if a correlation analysis reveals a connection between two characteristics.

Furthermore, statistical methods can help to determine whether data mining can actually lead to the desired result.

Visualization

Good *visualization* is essential for the success of a data mining project. Since data mining is mostly used for decision-making or decision support, and decisions are not always made by the people directly involved in the data mining process; the results of data mining must be illustrated. Only if the knowledge discovered can be presented in a clear and comprehensible manner will it be possible to generate confidence in the results and achieve acceptance of them.

Visualization can be used not only to present results, but also in the actual data mining process. Clever presentation of data often reveals first relationships between attributes. Examples include cluster formation or particularly influential attributes in a classification.

Visualization therefore not only represents the developed models graphically, but can also be used as a data mining technique in its own right in data analysis. Since the presentation of results is a decisive factor in the success of a project, a separate section is devoted to visualization, see → Section 9.6.

1.5 Why data mining?

Every introduction to the logical programming language PROLOG uses the example of family relationships, which could give the

impression that PROLOG is only used to describe family relationships. The situation is similar in data mining: the weather-game-yes-no example (see → Appendix A.3) is almost always used. You will also encounter this example repeatedly in this book, including in → Sections 1.6.3 and → 5.2. So, is data mining only useful for predicting games (will be played, will not be played)?

There are some successful examples from the early days of data mining that have since become classics:

- The American retail chain *Walmart* is said to have discovered that on certain days, diapers were sold particularly frequently together with beer. Although this example is often cited, there is still debate as to whether it is proven or belongs in the realm of legend.¹
- The credit check or assessment of creditworthiness of bank customers is an “old” application and is cited just as frequently [→ Han98].
- Personalized mailing campaigns as a marketing strategy are also found in many data mining introductions. Based on existing data, a model is developed for classification into the categories *write* and *do not write*, so that advertising information is only sent to potential customers.

Data mining has long since outgrown its infancy and has found its way into many fields of application: from finance to medicine, from customer analysis to e-learning. Applications and areas of application include, for example:

- Customer loyalty
- Customer relationship management
- Demand forecasts, for example for electrical energy
- Predictive maintenance
- Dynamic pricing

- Detection of fraudulent financial transactions, e.g., in online banking or with credit cards
- Diagnosis of illness
- Monitoring the health status of patients
- Detecting insurance fraud
- Detecting billing fraud, e.g., in healthcare

Examples of successful data mining applications are more likely to be found in older publications. Successful data mining is no longer anything special; it is no longer newsworthy.

[Hip+01] discusses a variety of successful applications in marketing. A typical application, customer segmentation, i.e., finding groups of similar customers, is presented in this book using several examples. The projects presented there come from the automotive sector and customized marketing. Successful projects from the second major application class, classification, are also presented, such as checking the creditworthiness and credit rating of customers. *Cross-selling*, as used by many online platforms, is also represented there with a successful project. Similar examples from the fields of e-business and finance can be found in [Soa+08]. In [→ Chu14], further data mining applications are addressed, including in particular data analysis in social networks.

Data mining techniques are also used in a variety of ways in e-learning. In [Rom+10] and [→ PA23], a number of ways in which data mining can be used to improve teaching are presented. These range from recognizing typical learning patterns, which enable user-customized presentation of content (so-called adaptive storyboards), to automatic detection of problems in the learning process. This field has become established under the term *Educational Data Mining* (EDM).

Successful data mining applications are, in turn, a good marketing argument for another group of companies: manufacturers of data mining software and cross-industry IT service companies in the field of data analysis depend on successful projects in order to attract new customers.

The website for KNIME, the freely available tool we use in this book, offers a number of suggestions. These are mainly introductory examples and give a good impression of the wide range of possible applications for data mining.

The list of customers who use *RapidMiner*, a tool that, like KNIME, is widely used, is long and covers many areas, from the electronics, aviation, and automotive industries to retail and market research companies, banks and insurance companies, the pharmaceutical and biotechnology industries, and the IT industry itself. Although specific application examples are not listed, the industries and companies mentioned provide indications of the use of data mining solutions.

For many years, tasks provided by companies were worked on in the Data Mining Cup. Several topics in the Data Mining Cup involve analyzing customer behavior and making predictions, whether for the use of vouchers or discount coupons, for book sales figures, or for customer behavior in a lottery.

In 2000 and 2001, a mailing campaign was the subject of the data mining task: Is it worthwhile to write to a customer or not? According to the organizers, both competitions were completed to the great satisfaction of the companies behind the task. In → Chapter 10, we revisit the topic from 2002, the mailing campaign of an energy supplier.

Successful data mining applications are integrated into the operational business of many areas of application and are used on a daily basis. Do your own research and be surprised by interesting applications of data mining.

1.6 Tools

The tasks covered in the book can be processed using both data mining systems and programs you have written yourself.

Regardless of this, knowledge of spreadsheet software is helpful. *Spreadsheet software* can be used to perform an initial analysis of the

data, for example to discover dependencies between attributes or analyze or rework the results of a data mining model.

A **text editor** that also allows you to work with macros is another useful tool for preparing and following up on data analysis.

Last but not least, it should be noted that it is sometimes necessary to develop small programs for data preprocessing and analysis evaluation. Any programming language can be used for this purpose. Some systems, including KNIME, allow you to incorporate your own Java or Python programs into the analysis process.

At this point, we provide a brief introduction to the data mining software we use to solve the tasks covered in the book:

- The Konstanz Information Miner (KNIME) is a system for describing entire data mining processes and includes a variety of algorithms for the different analysis phases. See [→ https://www.knime.com](https://www.knime.com).
- The Waikato Environment for Knowledge Analysis (WEKA) provides a set of algorithms implemented in Java that can be executed both interactively and in command line mode. See [→ https://ml.cms.waikato.ac.nz/weka/](https://ml.cms.waikato.ac.nz/weka/).
The WEKA algorithms can be integrated into KNIME as a KNIME extension.
- JAVANNS was developed in Stuttgart and Tübingen specifically for working with smaller neural networks. Unfortunately, JAVANNS is no longer being developed, but we still use it here for our introduction to neural networks because it provides excellent visualization of the networks and training. We provide JAVANNS on the book's website at [→ https://www.fww.hs-wismar.de/datamining/](https://www.fww.hs-wismar.de/datamining/).
- Python has become the preferred programming language for implementing data mining solutions. Powerful libraries for machine learning support the development of data mining models. We use the *Jupyter Notebook* programming

environment, which is included in the *Anaconda* installation:
→ <https://www.anaconda.com>.

There are other data mining tools, such as *RapidMiner*
(→ <https://rapidminer.com/>).

Data mining components are also integrated into many data warehouse and database systems.

In this book, we favor KNIME because it allows you to concentrate on developing a solution without having to deal with syntactic problems. The focus is on developing an adequate solution, not on enjoying a program that runs without errors.

1.6.1 Knime

KNIME is a data mining tool that was originally developed at the University of Konstanz. The abbreviation KNIME stands for *Konstanz Information Miner*². KNIME runs on all operating systems; *Java* is required. KNIME is available in different versions. In this book, we use the KNIME ANALYTICS PLATFORM.

KNIME is characterized by its very simple drag-and-drop operation and a large number of available algorithms and tasks. Its structure is modular and is constantly being expanded with new components that can be easily loaded internally via the program. In this way, the software can be expanded with a wide range of additional content in addition to its own algorithms. For example, almost all WEKA methods have been made available in KNIME. KNIME (in the freely available ANALYTICS PLATFORM version we use) is comparable in performance to a variety of commercial data mining programs. The entire mining process, from data import to data preprocessing and data analysis to the presentation of results, can be accomplished with the methods provided.

KNIME has numerous interfaces to other software packages and programming languages. In this book, we often use the WEKA tools integrated into KNIME. In addition, KNIME has interfaces to *R*, *Tableau*,

TensorFlow, Python, Keras, Perl, and others. Databases can also be integrated.

KNIME provides sample workflows that can be used for guidance. Some examples require the integration of WEKA methods into the KNIME system. WEKA methods can be integrated as KNIME extensions.

After starting KNIME, the window shown in → Figure 1.4 opens.

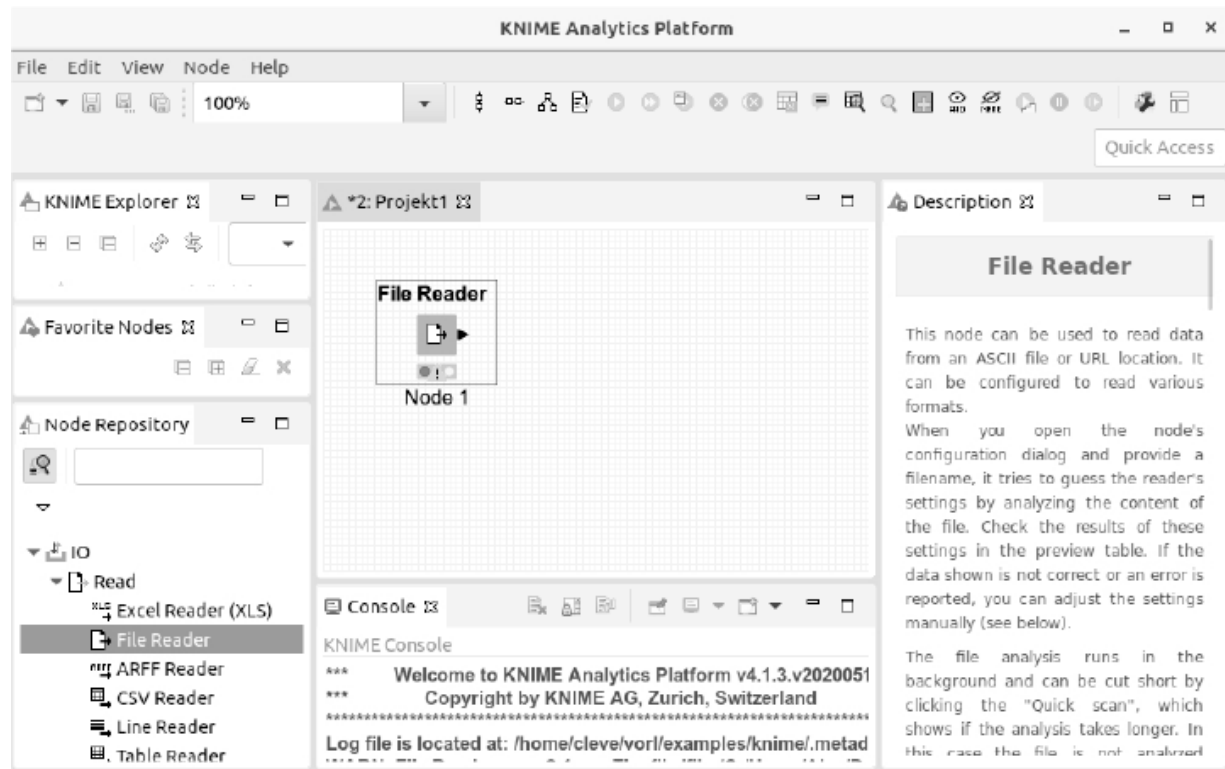


Fig. 1.4 KNIME workbench

The interface contains the following components:

1. Workflow window
The window in the middle is the main window, which displays the data mining process.
2. Project window
The *KNIME Explorer* project window is used to manage projects.

3. Node repository

The foundation of every KNIME application is the collection of implemented algorithms from which the user composes a workflow. The pool of algorithms includes components for the following tasks:

- Import and export from files or databases
- Data preprocessing and data manipulation
- Basic statistical analysis
- Data analysis in the narrower sense (both KNIME and WEKA)
- Visualization

4. In addition, there is the *Favorite Nodes* window, which displays frequently used nodes.

5. Node Description

This window provides a detailed description of the KNIME nodes:

- Description of the algorithm
- Required input formatting of data
- Type of the generated output
- Brief description of the configuration options

Now you can drag the corresponding tools as nodes from the repository into the main window and connect them there using arrows.

→ Figure 1.5 shows a workflow consisting of the following components.

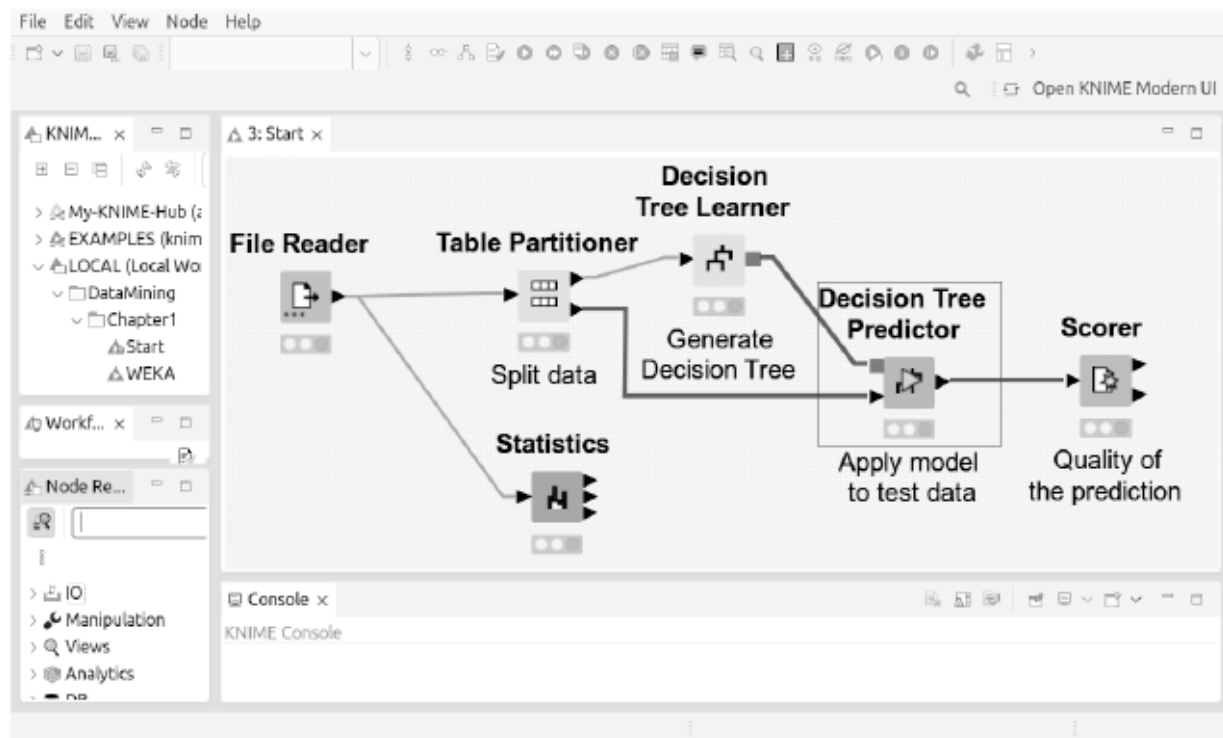


Fig. 1.5 Representation of a workflow in KNIME

1. File Reader: Here, the data is read in.
2. Partitioning: The data is divided into a training set and a test set (see → Section 9.4).
3. Decision Tree Learner: A decision tree is generated from the training set.
4. Decision Tree Predictor: The learned decision tree is applied to the test set.

The main window serves the following purposes:

- It is the primary modeling window of the mining process.
- The nodes can be managed and configured (right-click on the respective node).
- The nodes can be linked together to form a workflow for a data mining process. The arrows indicate the data flow.

Once a workflow has been assembled, the nodes must be configured. To do this, go to the respective node and open the configuration window by right-clicking the mouse (or double-clicking with the left mouse button). In the *File Reader*, you can select the file to be read in.

Once a node has been configured correctly, the traffic light below the node changes from red to yellow. Now execute the node, again by right-clicking. If everything has gone correctly, the traffic light will turn green. Continue this process for the subsequent nodes. If a warning triangle appears under the node, read the corresponding error or warning message and respond appropriately. → Figure 1.6 shows the finished workflow for the weather example (see → Appendix A.3).

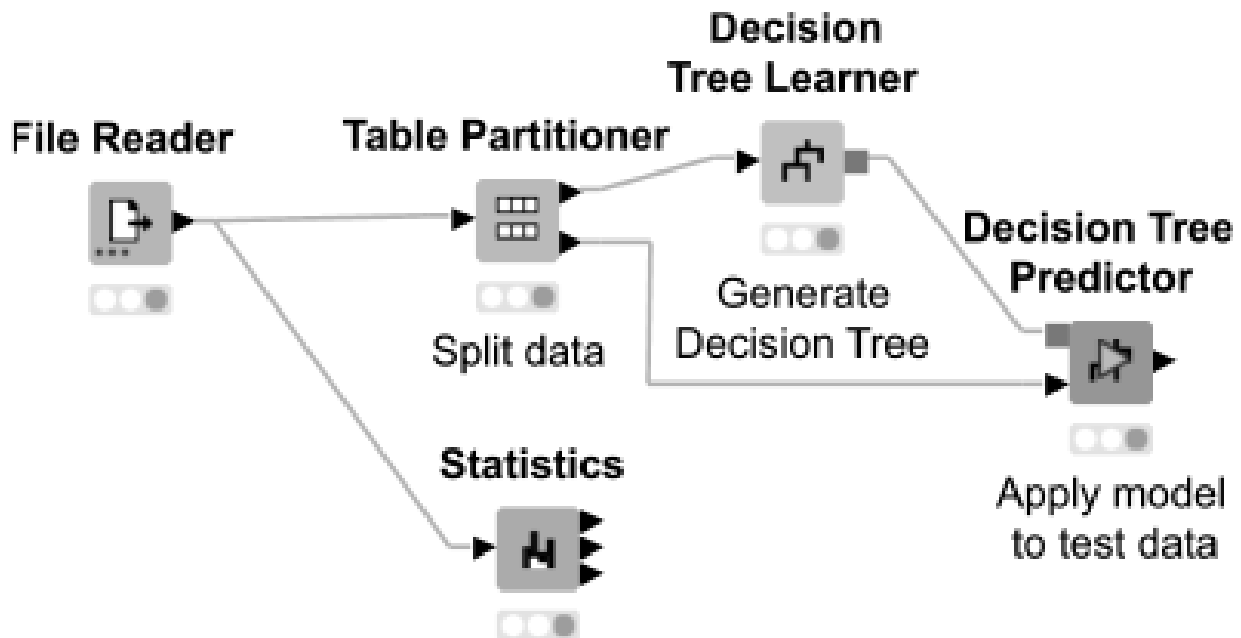


Fig. 1.6 Workflow for the weather example, representation in KNIME

The statistics node integrated into the workflow provides the statistical measures already mentioned and promotes understanding of the data to be analyzed.

Let's take another look at the key steps in the data mining workflow. First, the data must be imported. To do this, select the *File Reader* in the node selection under *IO* in the *Read* section and drag it onto the workspace. KNIME supports various formats, including csv, arff, and xls. The arff format is the standard format of the WEKA system.

KNIME contains a variety of preprocessing techniques. These can be found under *Data Manipulation*. The partitioner, for example, can be accessed under *Row > Transform*.

The actual analysis procedures are contained in the *Mining* folder. The *Decision Tree Learner* and *Decision Tree Predictor* nodes used in the above workflow are located in the *Decision Tree* subdirectory. In the *Decision Tree Learner*, you select which attribute should be the target attribute used for classification (*Class Column*). The configuration of the nodes often allows for many other settings, which we will not discuss here.

To check the hit rate of the decision tree during classification, the given sample set is divided into a training set and a test set. This is done by the *Partitioner* node. The training set is used to create a model, in this case a decision tree. We now apply this decision tree to the test set and observe how well or poorly it classifies the known data. Right-click on the *Decision Tree Predictor* and select *Classified Data*. The predictor inserts an additional column, *Prediction (DecTree)*, with the predicted classification for the respective data record (→ Figure 1.7).

Classified Data - 0:8 - Decision Tree Predictor (Baum auf)						
File						
Table "weather.numeric.csv" - Rows: 5			Spec - Columns: 6		Properties	Flow Variables
Row ID	S outlook	I tempe...	I humidity	S windy	S play	S Predic...
Row0	sunny	85	85	FALSE	no	yes
Row2	overcast	83	86	FALSE	yes	yes
Row4	rainy	68	80	FALSE	yes	yes
Row7	sunny	72	95	FALSE	no	yes
Row9	rainy	75	80	FALSE	yes	yes

Fig. 1.7 Representation of the result in KNIME

Alternatively, you can use the *Scorer* (→ Figure 1.8).

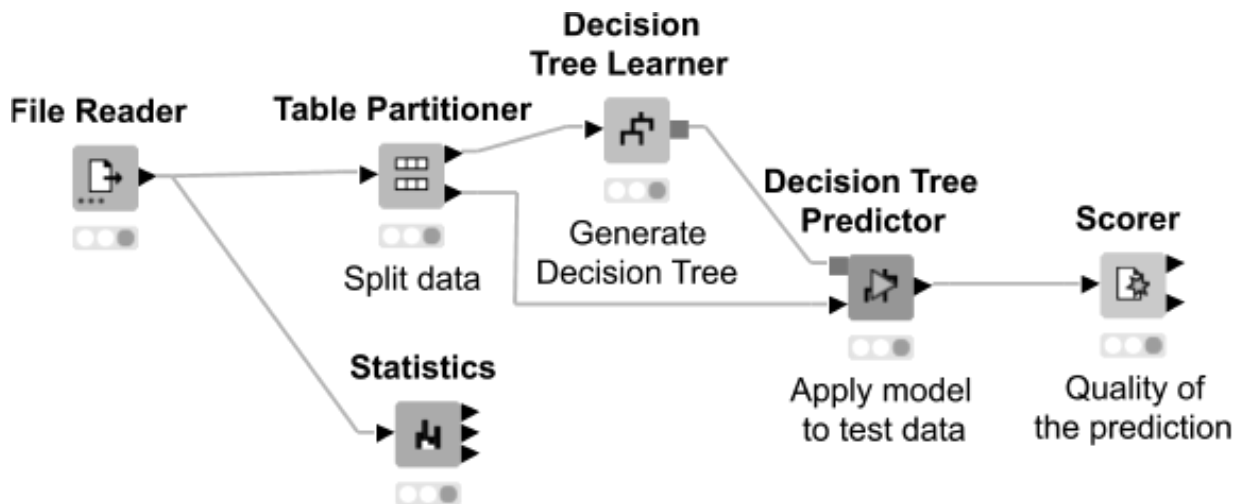


Fig. 1.8 The KNIME *Scorer*

This will give you evaluation scores (→ Figure 1.9), which we will discuss in → Chapter 9.

Accuracy statistics - 3:12 - Scorer (Quality)									
File Edit Hilit Navigation View									
Table "default" - Rows: 3 Spec - Columns: 11 Properties Flow Variables									
Row ID	I TrueP...	I FalseP...	I TrueN...	I False...	D Recall	D Precisi..	D Sensit...	D Specifi...	D F-meas...
no	1	1	2	1	0.5	0.5	0.5	0.667	0.5
yes	2	1	1	1	0.667	0.667	0.667	0.5	0.667

Fig. 1.9 KNIME scores

Incidentally, the method of calculating the scores shown here can be simplified by combining the *Scorer* and *Math Formula* nodes, as described in → Chapter 10.

We would like to point out two options that KNIME offers in addition to its large stock of ready-made nodes. First, it is possible to develop custom nodes.

→ Figure 1.10 shows a so-called *Java snippet* node. The *Java snippet* node calculates the points for the Data Mining Cup 2007 task. The aim is to predict whether a customer will redeem coupon A, coupon B, or no coupon (N). The correct prediction of B is awarded 6 points, the correct prediction of A is awarded 3 points, and N is awarded 0 points. An incorrect prediction of A or B is penalized with -1. The Java snippet node calculates how many points are awarded for each data record. To do this, the *points* variable is first declared and initialized with 0. If the values for the attributes *pred*, the prediction column *Prediction (DecTree)* has been renamed, and *COUPON* (the actual behavior of the customer) are the same, 3 or 6 points are awarded if the prediction is A or B, respectively. Similarly, incorrect A/B predictions are penalized with -1. A new column, *Points*, is added. Each data record then has a new value, *Points*.

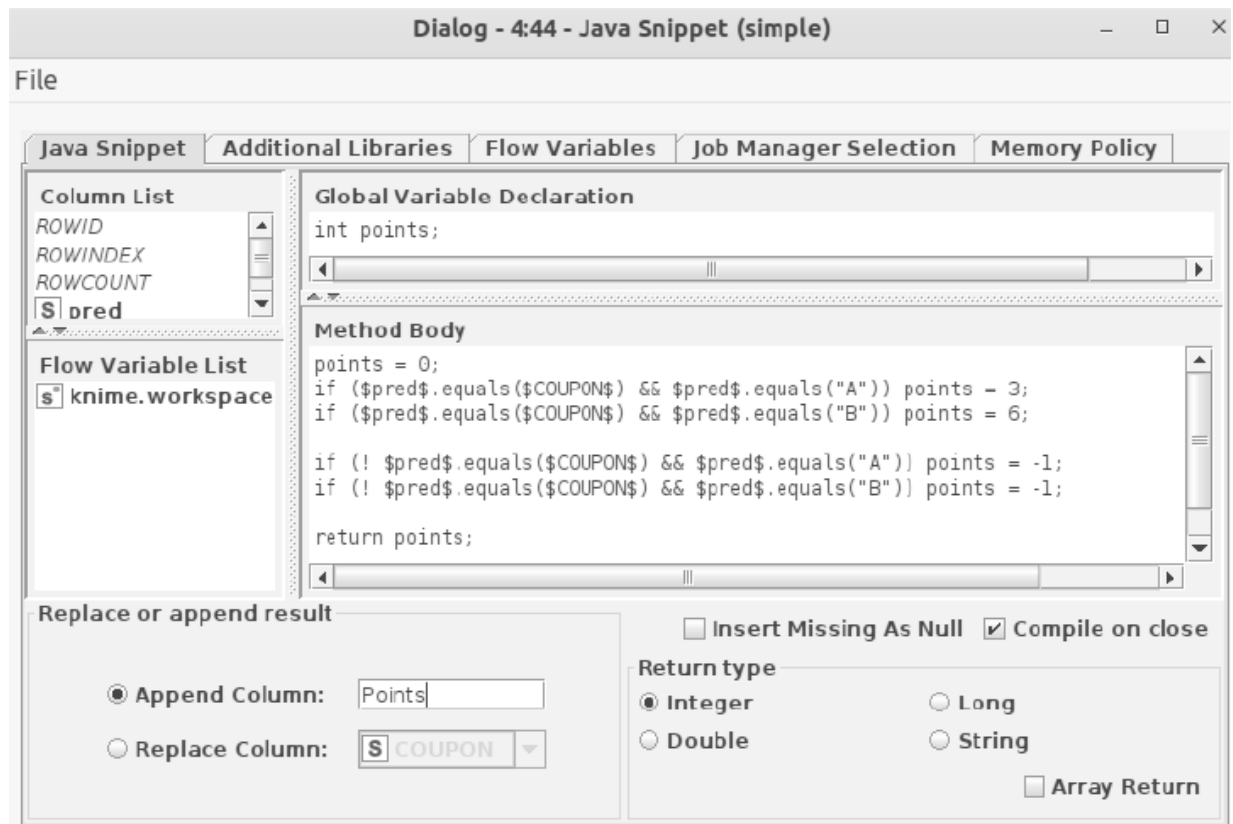


Fig. 1.10 Java snippet for scoring

In the next Java snippet (→ Figure 1.11), the points are now added up. A new column, *GSum*, is created, which computes the sum of all points up to that data record for each data record. The entry for the last data record is then the total number of points achieved.

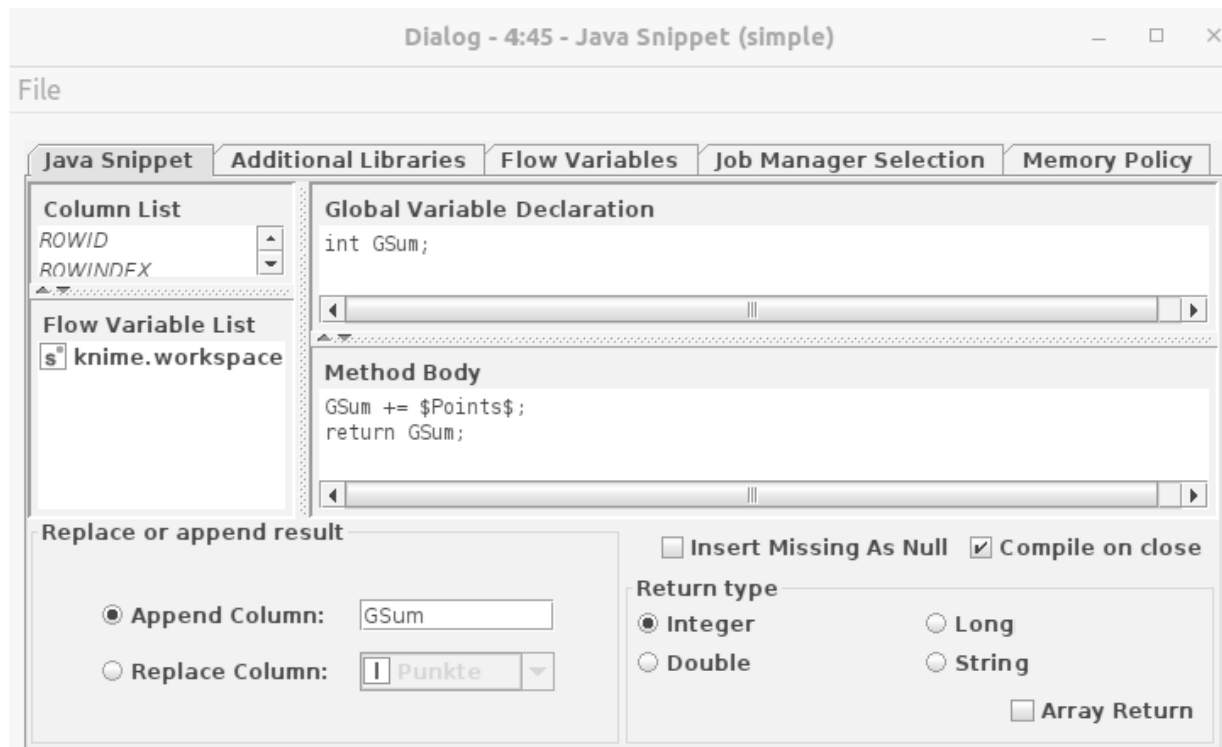


Fig. 1.11 Java snippet for summing points

A second helpful feature offered by KNIME is loops, which can be used to define multiple passes through a workflow. We will start with a simple workflow (→ Figure 1.12) for classifying the data from the Iris example (see → Appendix A.1 and Example 6.1) using the k-nearest neighbor method (see → Section 5.1). The goal is to correctly predict the Iris species based on the given measurements of each plant.



Fig. 1.12 KNIME, loops

The data is divided in a ratio of 80 % to 20 % in such a way that the frequency distribution of the class values in the subsets is equal to the distribution in the total set (*stratified sampling*, see → Section 9.4).

We then predict the class of the data records in the 20 % set by selecting the $k=3$ most similar data records from the 80 % set and determining the most frequent class there.

However, choosing $k=3$ for the nearest neighbors, on the basis of which we predict the class, is quite arbitrary. We would prefer to be able to perform the calculations with *different* values of k . It is precisely this trial and error with different values for k that is now implemented using the loop nodes. First, we display the loop nodes in the *K Nearest Neighbor* node (right mouse button, *Show Flow Variable Ports*). Then we drag the *Table Creator* and the *TableRow To Variable Loop Start*, as well as the *Loop End* node into our workflow (→ Figure 1.13).

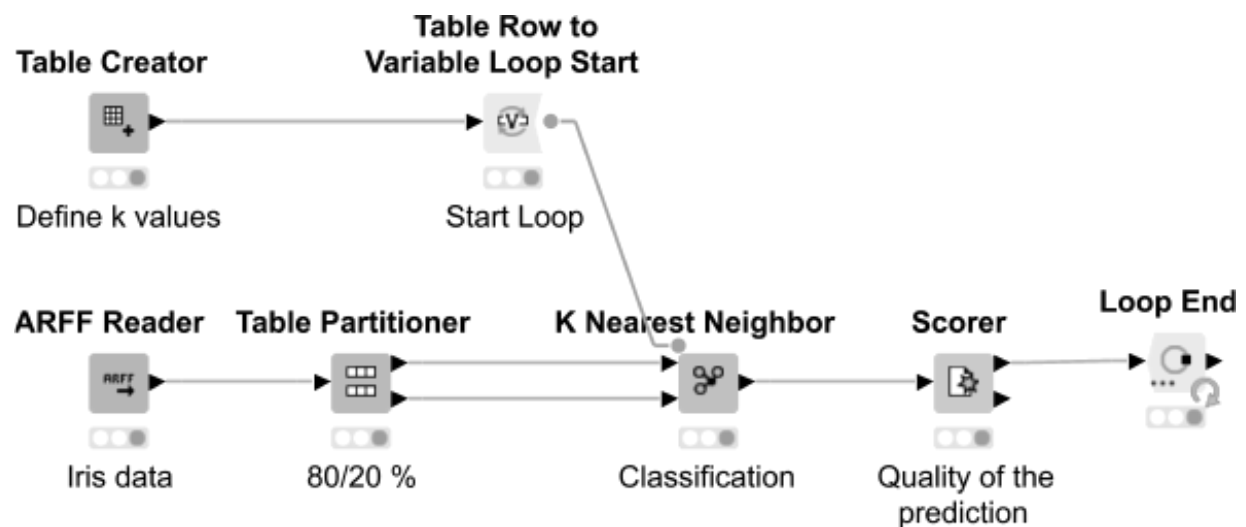


Fig. 1.13 Using loops in KNIME, advanced workflow

In the *Table Creator*, we define the variants for k (→ Figure 1.14).

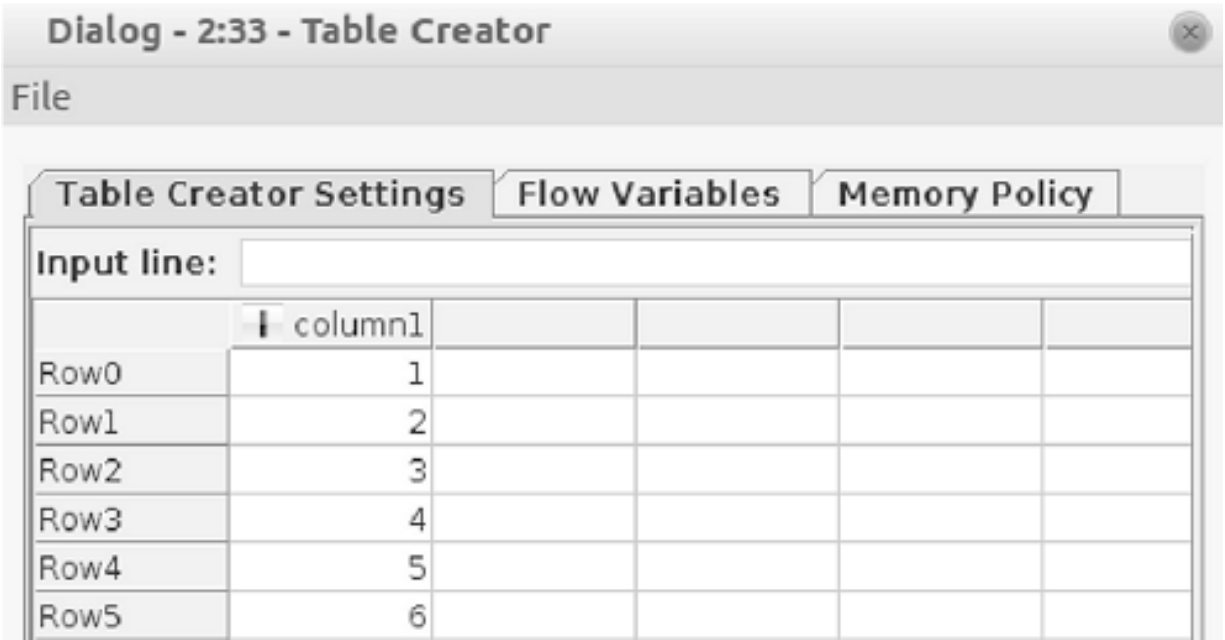


Fig. 1.14 Using loops in KNIME, Table Creator

In the *K Nearest Neighbor* node, it is now specified that the numbers from the *Table Creator* should be used for *k* (→ Figure 1.15).

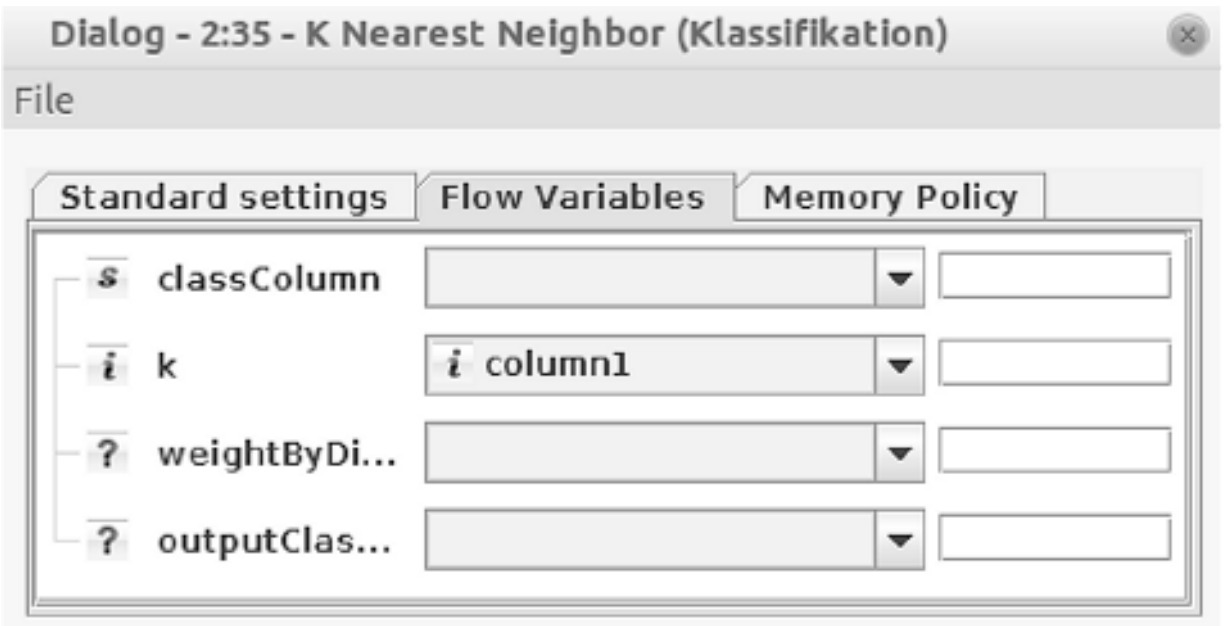


Fig. 1.15 Using loops in KNIME, k-nearest neighbor dialog

The log that we display at the end of the loop is shown in → Table 1.1. The Iteration column does not show the selected k , but rather the variant number, with counting starting at 0.

Tab. 1.1 Iris data with k-nearest neighbor(loop)

row ID	Iris-setosa	Iris-versicolor	Iris-virginica	Iteration
Iris-setosa#0	10	0	0	0
Iris-versicolor#0	0	10	0	0
Iris-virginica#0	0	1	9	0
Iris-setosa#1	10	0	0	1
Iris-versicolor#1	0	10	0	1
Iris-virginica#1	0	1	9	1
Iris-setosa#2	10	0	0	2
Iris-versicolor#2	0	10	0	2
Iris-virginica#2	0	1	9	2
Iris-setosa#3	10	0	0	3
Iris-versicolor#3	0	10	0	3
Iris-virginica#3	0	1	9	3
Iris-setosa#4	10	0	0	4
Iris-versicolor#4	0	10	0	4
Iris-virginica#4	0	1	9	4
Iris-setosa#5	10	0	0	5
Iris-versicolor#5	0	10	0	5
Iris-virginica#5	0	1	9	5

It is noticeable that the same result is achieved with all selected k values. The prediction is always correct except for one data record.

1.6.2 Weka

WEKA is a freely available data mining tool for non-commercial use, developed at the University of Waikato in New Zealand. WEKA is an abbreviation for *Waikato Environment for Knowledge Analysis*. Many

algorithms are implemented in WEKA, see [Wit+25] for more information. WEKA is implemented in *Java* and therefore runs on all operating systems. You can find WEKA along with additional information at [→ https://ml.cms.waikato.ac.nz/weka/](https://ml.cms.waikato.ac.nz/weka/).

WEKA is started via `java -jar weka.jar` in a command line interface or by double-clicking on the jar file. The window shown in → Figure 1.16 opens.



Fig. 1.16 WEKA

We work with the *Explorer*, but the experimentation environment (*Experimenter*) and a command line window can also be opened in the WEKA start window. The WEKA Explorer opens in a new window (→ Figure 1.17), which contains a kind of tabbed table with the tabs *Preprocess*, *Classify*, *Cluster*, *Associate*, *Select attributes*, and *Visualize*.

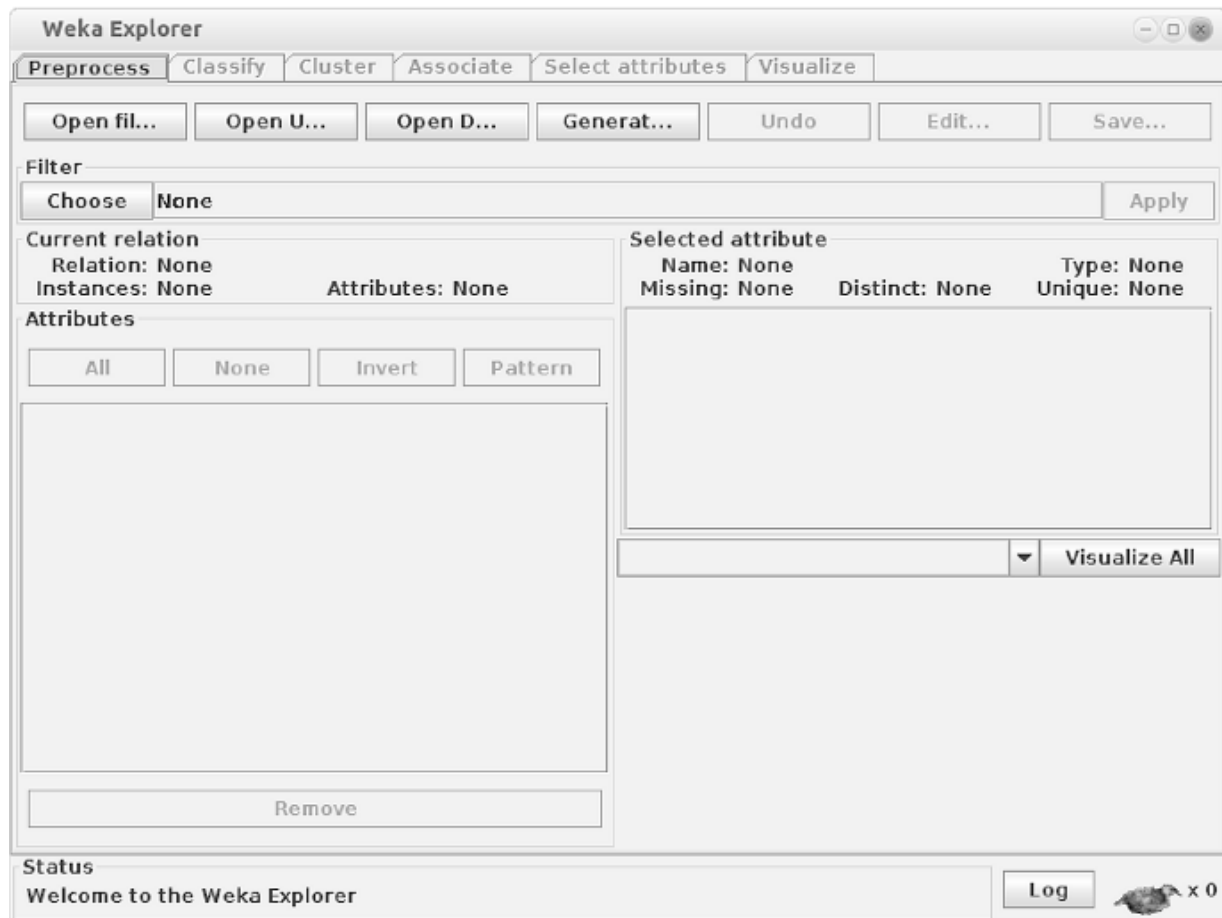


Fig. 1.17 WEKA Explorer

In the *Preprocess* tab, the settings for the data sets to be used can be configured. Data sets can be loaded into the WEKA system from a file (1), via a URL (2), or from a database (3) (→ Figure 1.17). Files are expected in WEKA in ARFF format (→ Example 1.1). ARFF files consist of a header and a body. The header contains keywords beginning with @, such as *@relation* and *@attribute*. Only nominal and numerical data are recognized. For nominal attributes, the value set follows in curly brackets {...}, for numeric attributes, the keyword numeric or real follows. Lines beginning with % are comments. The data is contained in the body (from *@data*). Each data record is on a separate line, with the individual attributes separated by commas. A question mark "?" denotes a missing value.

Example 1.1 (Weather data in WEKA).

```
@relation weather.symbolic
@attribute outlook {sunny, overcast, rainy}
@attribute temperature {hot, mild, cool}
@attribute humidity {high, normal}
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}
@data
sunny, hot, high, FALSE, no
sunny, hot, high, TRUE, no
overcast, hot, high, FALSE, yes
rainy, mild, high, FALSE, yes
rainy, cool, normal, FALSE, yes
rainy, cool, normal, TRUE, no
overcast, cool, normal, TRUE, yes
sunny, mild, high, FALSE, no
sunny, cool, normal, FALSE, yes
rainy, mild, normal, FALSE, yes
sunny, mild, normal, TRUE, yes
overcast, mild, high, TRUE, yes
overcast, hot, normal, FALSE, yes
rainy, mild, high, TRUE, no
```

When this file is loaded into WEKA (→ Figure 1.18), the attribute names appear.

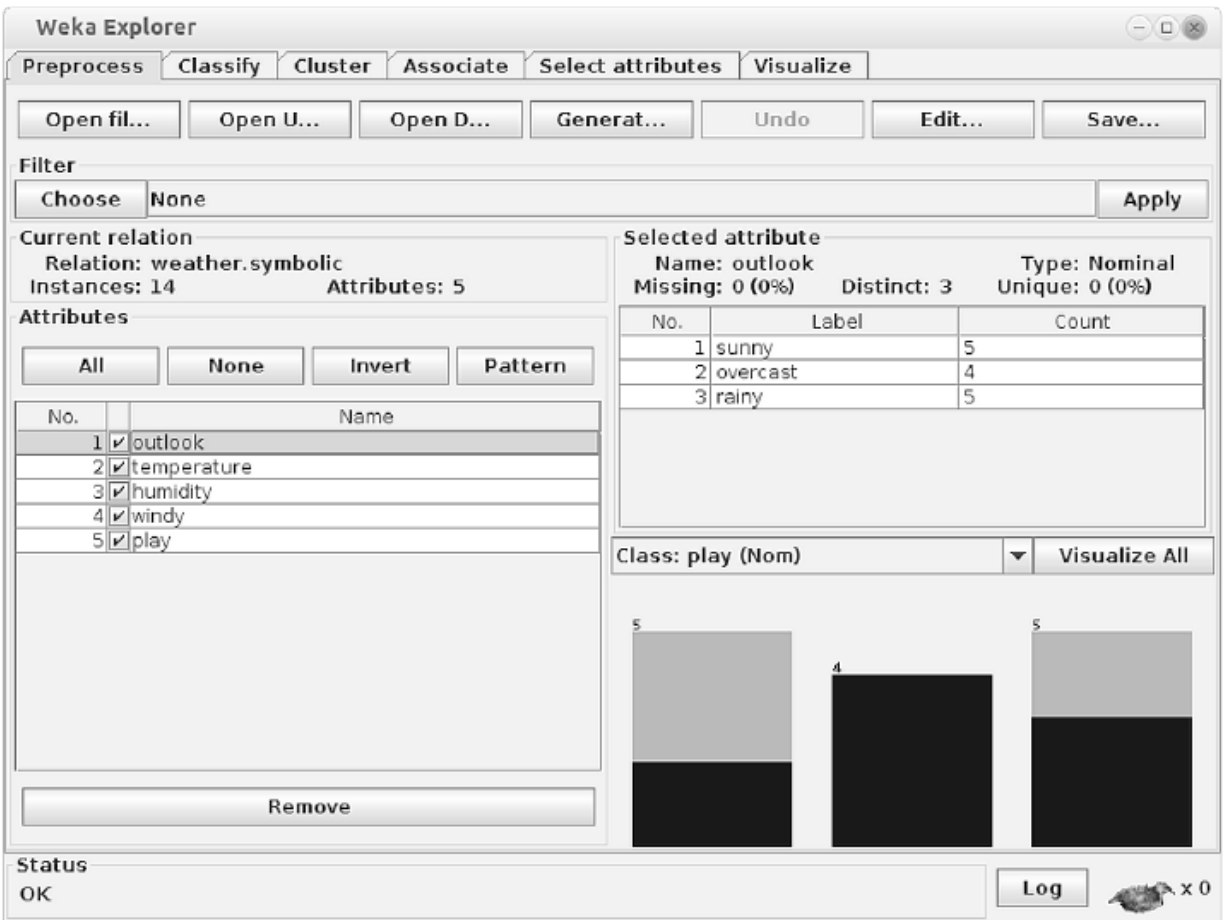


Fig. 1.18 Preprocessing in WEKA

Now the attributes to be used in the further process can be selected. The lower right part shows the distribution of the target attribute *play* with respect to the attribute selected in the left part of the window.

WEKA offers filters that can be used to preprocess the data. Several classifiers are implemented in the *Classify* program section. If the ID3 algorithm is selected (under *Choose > Trees*) and started, the result of the calculation is displayed in the right-hand panel (→ Figure 1.19).

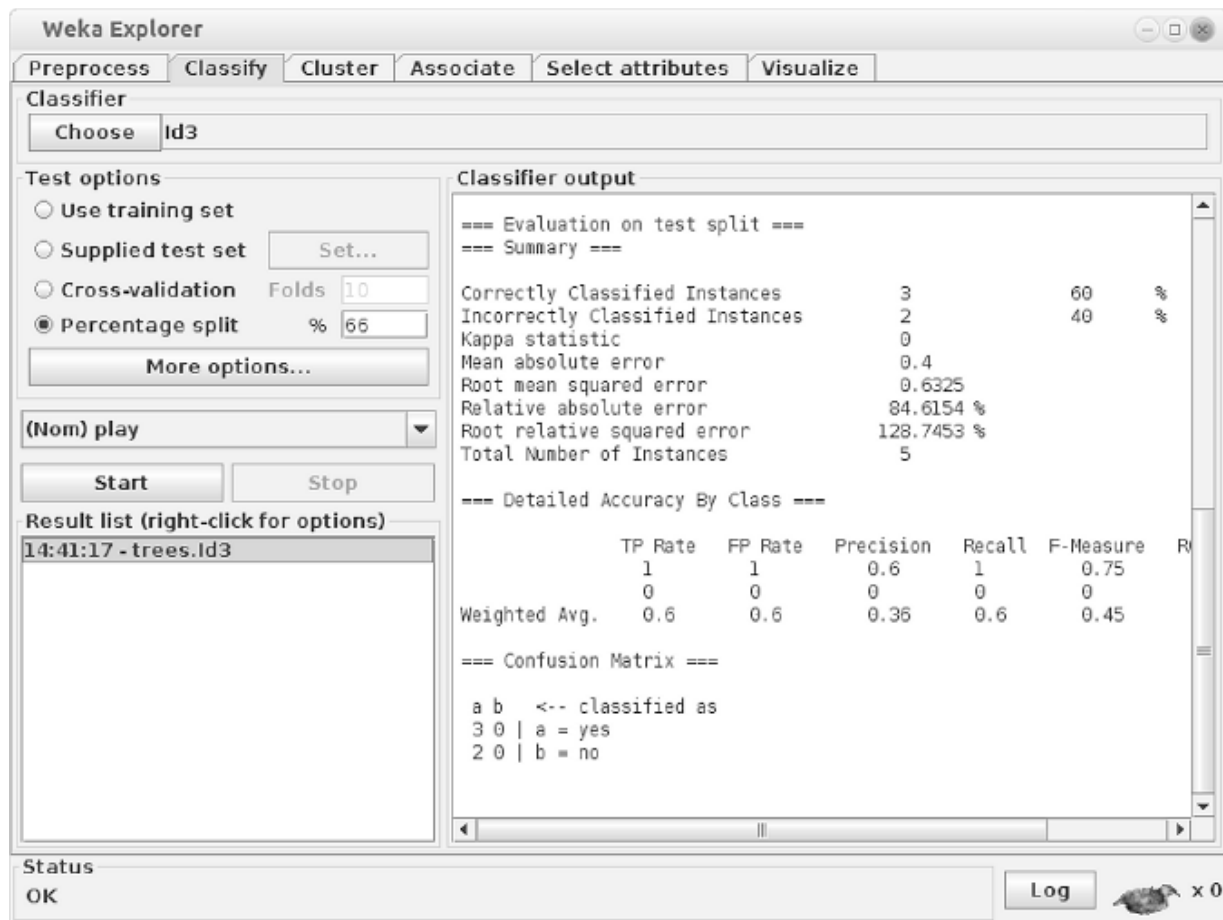


Fig. 1.19 Using ID3 in WEKA

The data to be learned can be selected:

Use training set:	The entire data set is used for learning.
Supplied test set:	A separate file can be specified for learning.
Cross validation:	The given set is split using cross validation (see → Section 9.4).
Percentage split:	The given set is split into a training set and a test set on a percentage basis.

For the J48 algorithm, a variant of the extension of the ID3 algorithm C4.5, a visualization of the decision tree (→ Figure 1.20) is available (right-click on the result, *Weka Node View*).

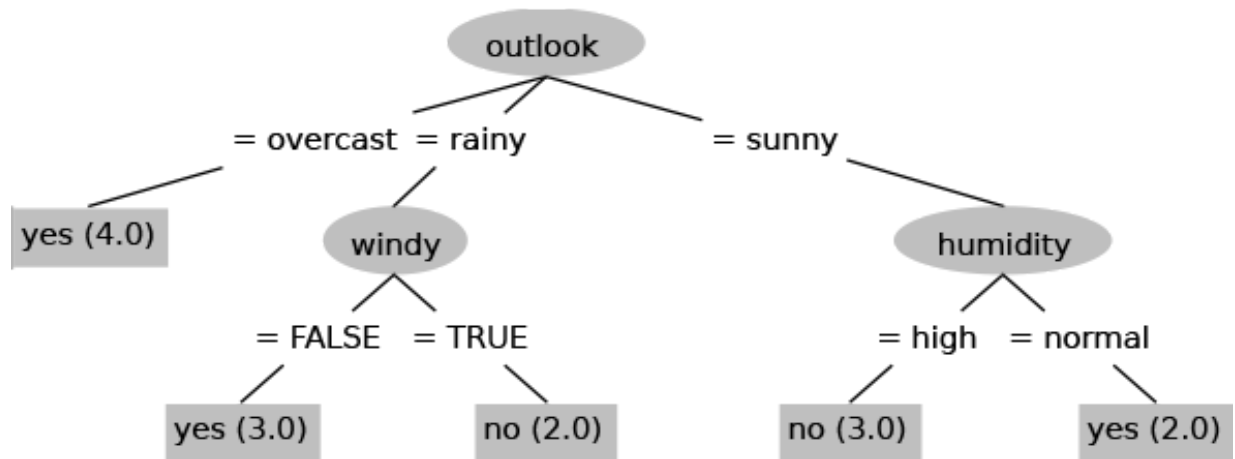


Fig. 1.20 A decision tree in WEKA

The procedure is similar for the other application classes. To calculate clusters, select the *Cluster* tab. Then select, for example, the *SimpleKMeans* algorithm (→ Section 6.2) under *Choose*. We cluster our entire sample set (*Use training set*). The result is shown in → Figure 1.21.

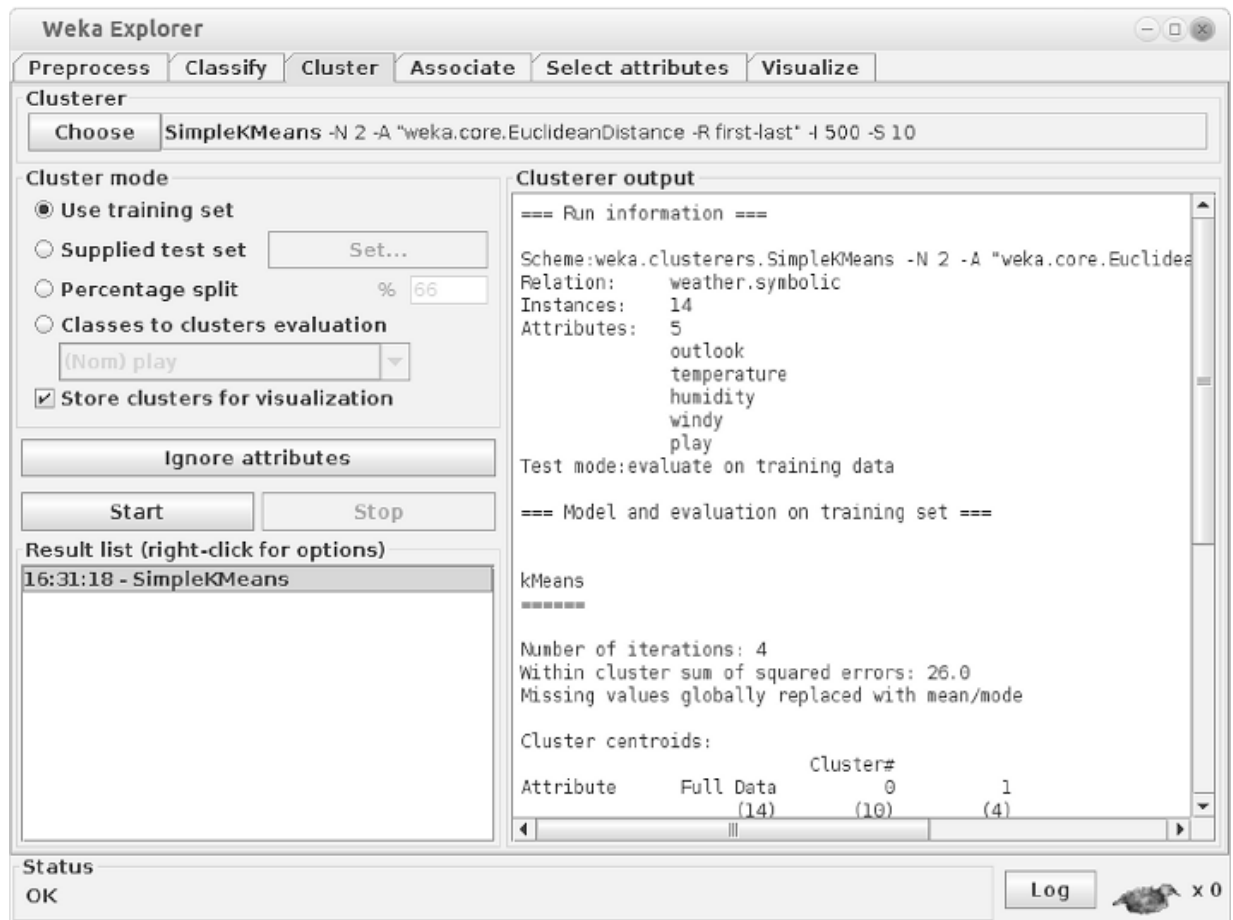


Fig. 1.21 Using SimpleKMeans in WEKA

1.6.3 JAVANNS

In the 1990s, the Stuttgart *Neural Network Simulator* (SNNS) was developed at the University of Stuttgart and later equipped with a user interface implemented in *Java*, the JAVANNS. The JAVANNS is an excellent tool for learning the principles of artificial neural networks. In particular, its visualization capabilities help to understand both architectures and learning processes, enabling them to be used effectively in data mining.³

JAVANNS can be used to develop various types of artificial neural networks. We will explain how to work with JAVANNS by developing a feedforward artificial neural network for a classification task. The

weather problem (see → Appendix A.3) serves as an example: Will the game be played? The data can be found in → Table 1.2.

To work with a neural network, ordinal data must be transformed into a binary representation. The neural network component in KNIME automatically converts such data, treating ordinal attributes in the same way as nominal attributes. An attribute value is encoded by a 0-1 sequence that has as many digits as the attribute has possible values.

Tab. 1.2 Weather data

Day	Outlook	Temperature	Humidity	Windy	Play
1	sunny	hot	high	false	no
2	sunny	hot	high	true	no
3	overcast	hot	high	false	yes
4	rainy	mild	high	false	yes
5	rainy	cool	normal	false	yes
6	rainy	cool	normal	true	no
7	overcast	cool	normal	true	yes
8	sunny	mild	high	false	no
9	sunny	cool	normal	false	yes
10	rainy	mild	normal	false	yes
11	sunny	mild	normal	true	yes
12	overcast	mild	high	true	yes
13	overcast	hot	normal	false	yes
14	rainy	mild	high	true	no

For example, the values of the *outlook* attribute are encoded as follows:

rainy	= (1, 0, 0),
overcast	= (0, 1, 0),
sunny	= (0, 0, 1).

However, since this results in a loss of order (*sunny* > *overcast* > *rainy*) we use the following coding for working with JAVANNNS, which

preserves the order.

outlook:	rainy = (0, 0);	overcast = (0, 1);	sunny= (1, 1);
temperature:	cold = (0, 0);	mild = (0, 1);	hot = (1, 1);
humidity:	normal= 0;	high = 1;	
windy:	false = 0;	true = 1;	
play:	no = 0;	yes = 1;	

To train the network, a pattern file (*.pat*) must be created that contains all patterns in coded form and has the following structure:

```
SNNS pattern definition file V3.2
generated at Mon Apr 26 18:08:50 2021
No. of patterns      : 14
No. of input units   : 6
No. of output units  : 1
```

```
# Input pattern 1:
1 1  1 1  1  0
# Output pattern 1:
0
# Input pattern 2:
1 1  1 1  1  1
# Output pattern 2:
0
# Input pattern 3:
1 1  0 1  1  0
# Output pattern 3:
0
...
```

Starting with the coding has the advantage that it determines both the size of the input layer and that of the output layer. For this example, a network with six input neurons and one output neuron is required. We will add a small hidden layer consisting of two

neurons. To do this, the following steps must be performed after starting JAVANNS, see → Figure 1.22.

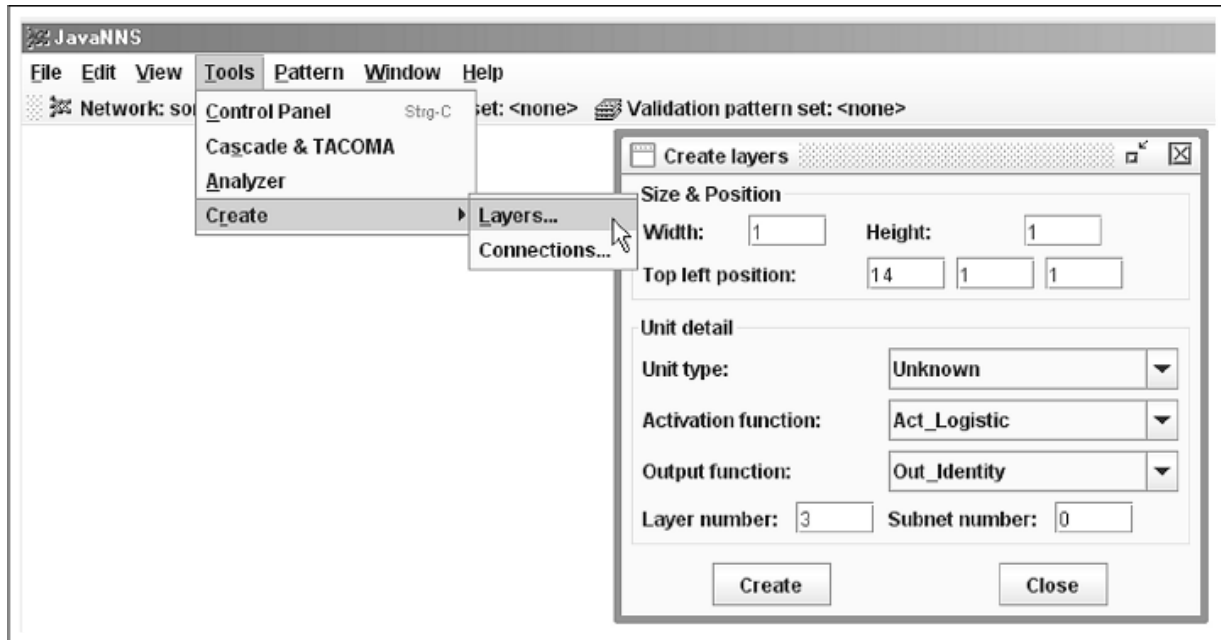


Fig. 1.22 JAVANNS: Creating a neuron layer in JAVANNS

- Create the three neuron layers via *Tools > Create > Layers...*
- The first layer of 6 neurons can be created as a 2x3 or 1x6 layer in the *Create Layer* window. The type of neurons (*Unit Type*) must be set to *Input*.
- The hidden layer is designed as a 1x2 layer and the type is set to *Hidden*. Change the *top-left position* of the layer by increasing the recommended value by 1 so that there is a visible gap between the created neuron layers in the representation.
- The output layer consists of only one neuron (1x1 layer). Change the *top-left position* as specified above and set the type of the neuron to *Output*.
- Establish the feed-forward connection between the layers: *Tools > Create > Connection*

Select *Connect feed-forward* and don't forget to create the connections using *Connect*.

- Save the network.
- Load the sample file. This file is a *.pat* file and must contain the content shown above.

The network can now be “embellished” by giving the input and output neurons appropriate names. To do this, select a neuron and edit it accordingly. Now the artificial neural network can be trained to generate a classifier. This is done by opening the *control* window (*Tools > Control*). To check the learning success, it is useful to display the network error via *View > Error graph*.

The training environment is shown in → Figure 1.23.

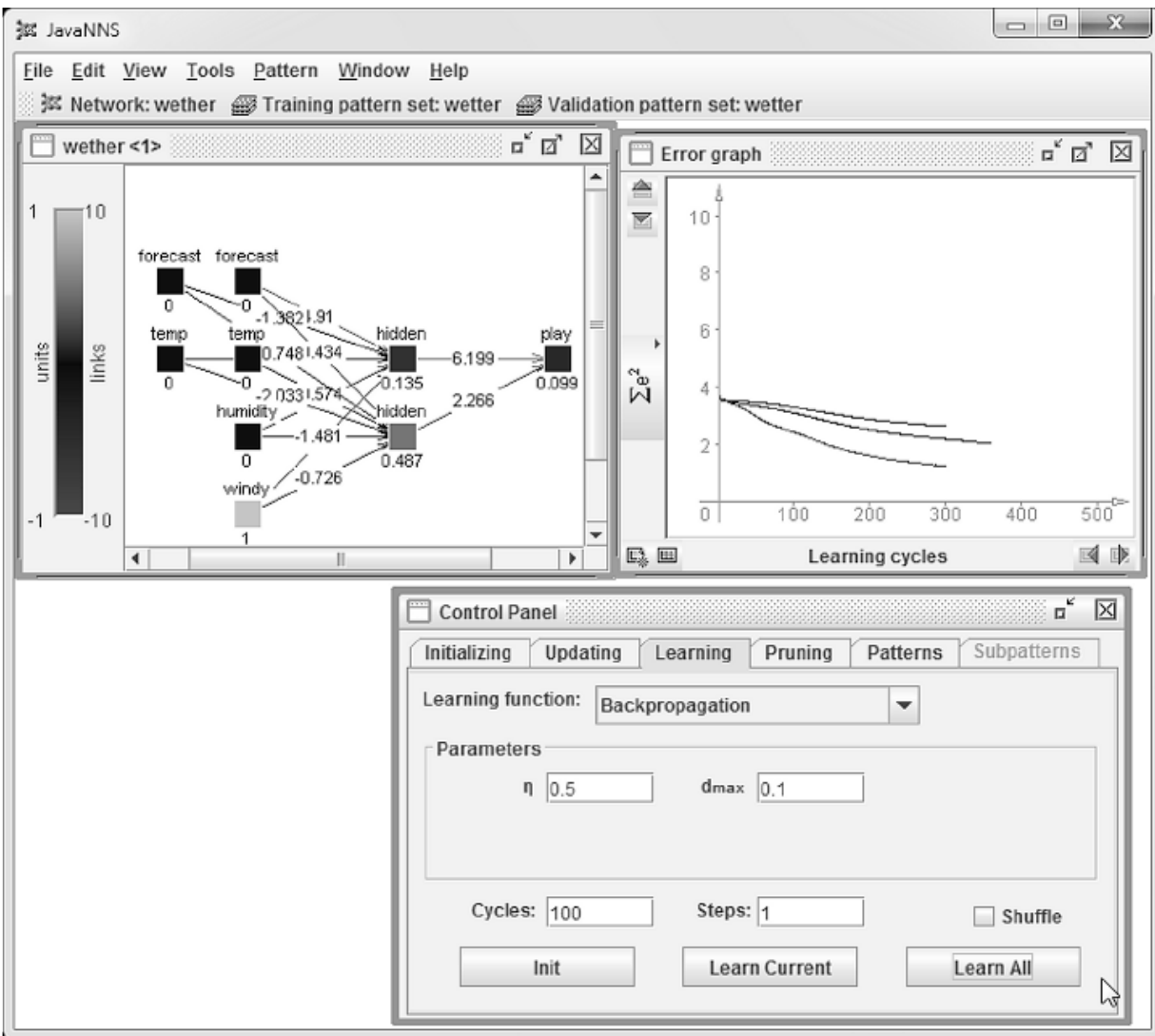


Fig. 1.23 Training a network in JAVANNS

Once the network has been initialized, it can be trained using the default values (*back-propagation learning method, learning parameter 0.2*). To do this, the number of cycles is specified and training is started using *Learn All*. The error curve gives an impression of the learning success.

The result can then be saved using *Save Data > Select Result files .res* (→ Figure 1.24).

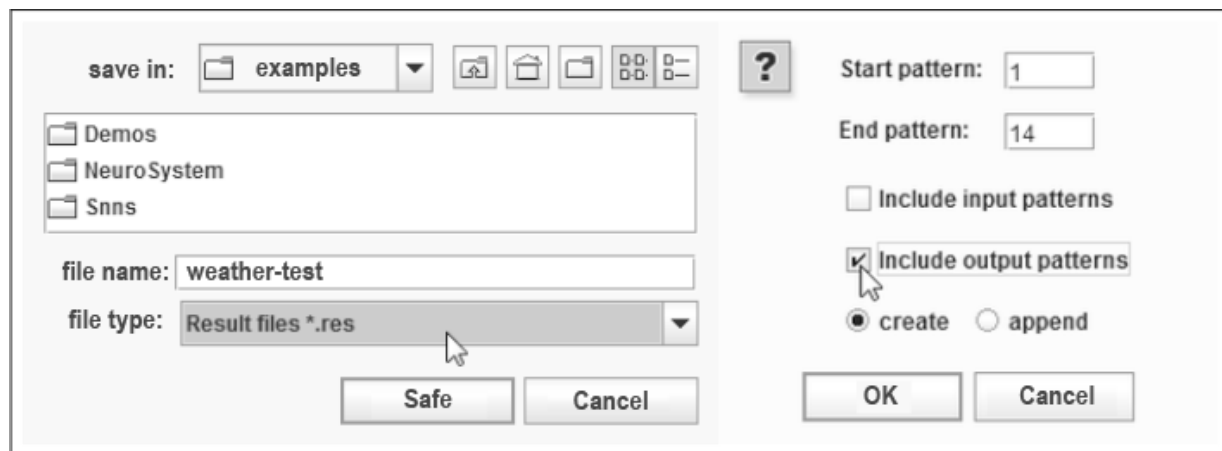


Fig. 1.24 Saving the result in JAVANNS

The training output (see → Figure 1.24, *Include output patterns*) is included in the result data. This creates a text file that contains both the training value and the value calculated by the network for each pattern, allowing further evaluation, for example using a spreadsheet. The following listing shows the contents of a *.res* file.

```
SNNS result file V1.4-3D
generated at Sun May 03 16:43:13 2021
No. of patterns      : 14
No. of input units   : 6
No. of output units  : 1
startpattern         : 1
endpattern           : 14
teaching output included
#1.1
0
0.20123
#2.1
0
0.01936
#3.1
```

```
0
0.9002
#4.1
1
0.78917
#5.1
1
0.94537
```

JAVANNNS can be used for many types of artificial neural networks, but is preferably used for classification tasks using feedforward neural networks.

1.6.4 Python

The Python programming language has become the number one choice for programming machine learning solutions. We are not offering an introductory course in Python programming here, but rather presenting the development environment we use to work on data mining problems. We use the *Jupyter Notebook* programming environment, which is provided by *Anaconda*⁴ The versions available at the time of writing are:

- Anaconda Navigator 2.7.0
- Jupyter Notebook 7.2.2
- Python 3.12.8

The installation of Anaconda also includes the Jupyter Notebook and Python.

Jupyter Notebook conveniently combines *Python* programming with interactive execution of program sections and their documentation. The heading in → Figure 1.25 is a so-called *Markdown* cell, which can be used to describe a program solution in detail beyond the *Python* comments. The figure shows cells with Python code and the result of executing a cell.

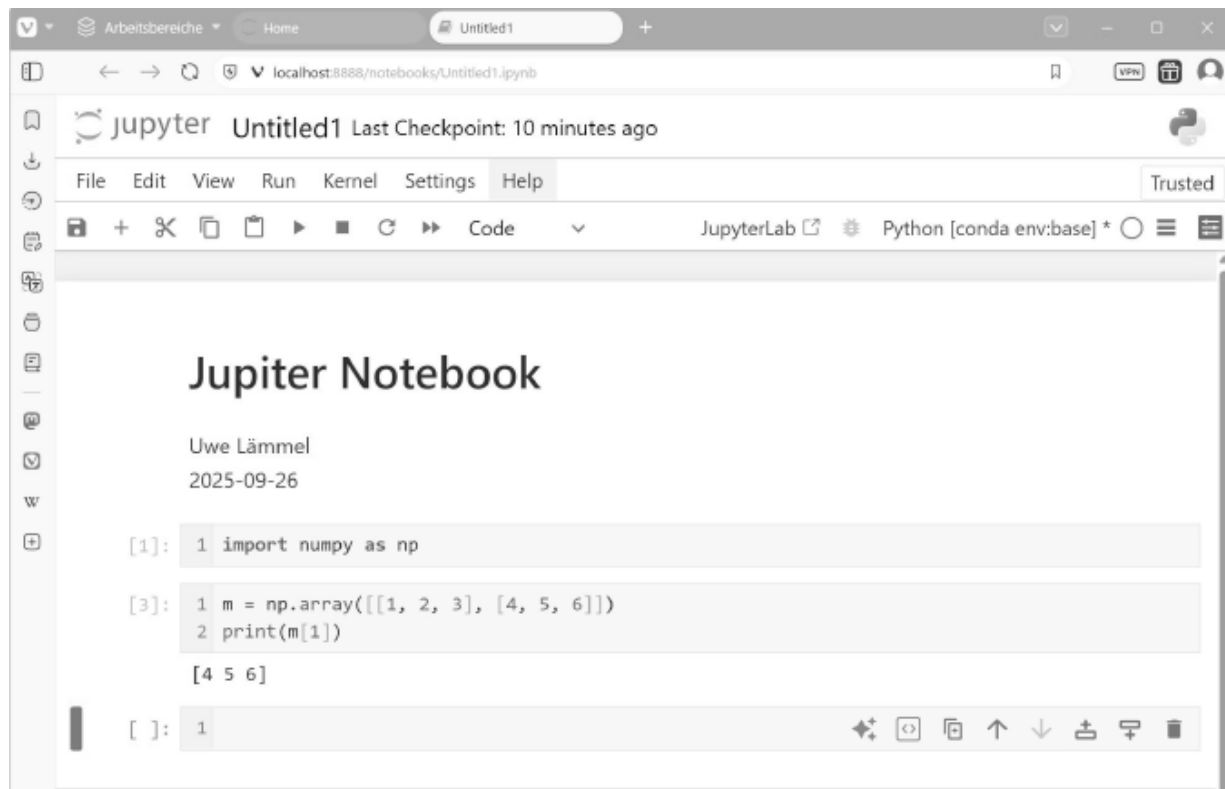


Fig. 1.25 Jupyter Notebook with Markdown cell, Python cell, and output

Powerful open-source packages have been developed for dealing with machine learning problems:

Keras can be seen as part of TensorFlow. It provides the interface to TensorFlow and is thus used for the development and evaluation of neural networks.

→ <https://www.tensorflow.org/guide/keras>

Matplotlib provides functions for visualization. In machine learning, for example, frequency distributions of attributes or error curves in model training are often displayed graphically. Here is the Python code for drawing the logistic function (→ Figure 1.26), which is used as an activation function in neural networks:

```
# Logistic activation function
```



```
# A Matplotlib-example
import matplotlib.pyplot as plt
import numpy as np
```

```
x = np.linspace(-2, 2, 500)
y = 1/(1+np.exp(-4*x))
```

```
figure, axes = plt.subplots()
axes.plot(x, y)
plt.show()
```

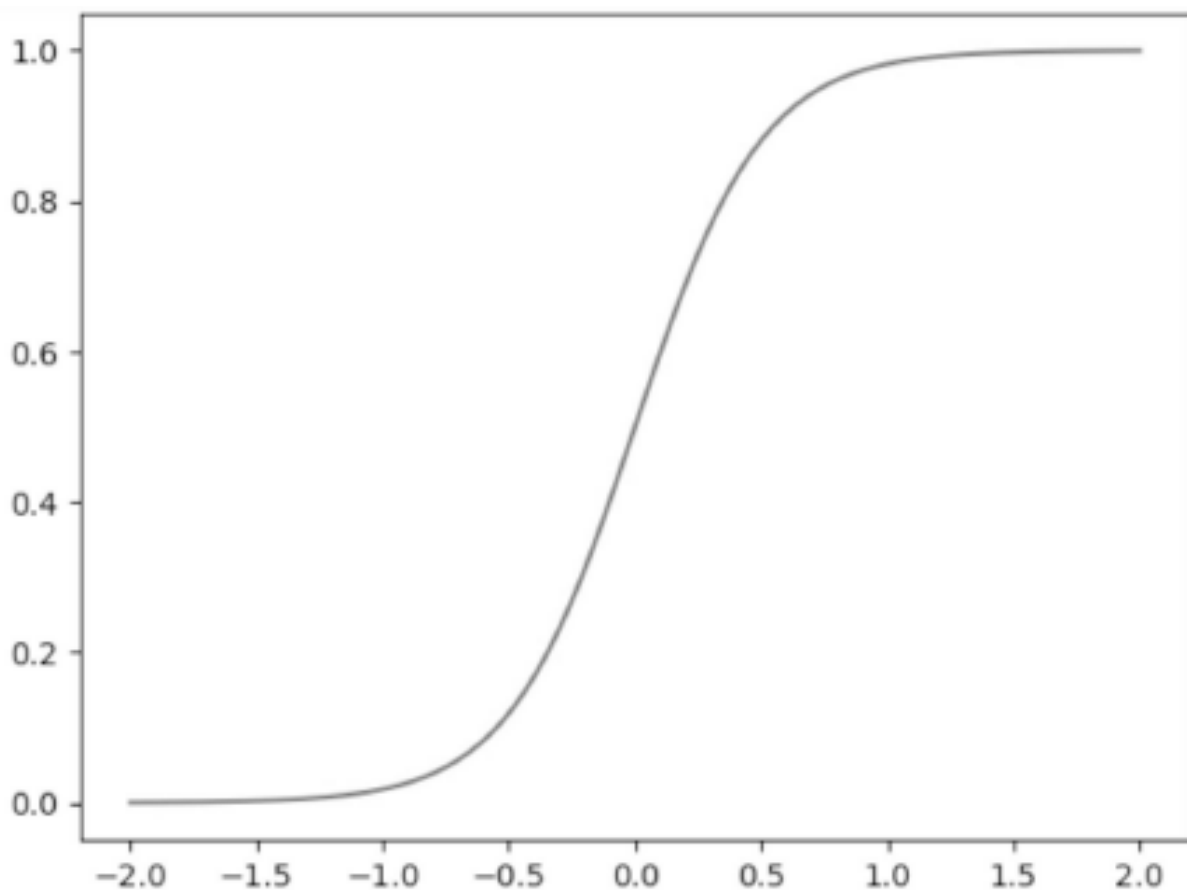


Fig. 1.26 Logistic function plotted with Matplotlib

NumPy is the basic package for scientific calculations. At its core is the *numpy.ndarray* object. An *ndarray* is a multidimensional array of fixed size, where all elements are of the same type. *NumPy* provides powerful and easy-to-use operations for input and output as well as for calculations on n-dimensional matrices. Many operations are programmed in C and are available in compiled form, allowing calculations to be performed very efficiently. → <https://numpy.org>

Pandas offers an efficient way to manage and examine data with a table structure using the *DataFrame* object. This makes it easy to input and output data sets with table headers or to change the structure using the column labels (usually the attribute names).

→ <https://pandas.pydata.org>

Scikit-learn is a library for machine learning. It provides functions for supervised and unsupervised learning. It supports a whole range of data mining methods and provides tools for preprocessing and model evaluation. → <https://scikit-learn.org/stable/>

Scipy builds on the *NumPy* package to provide powerful operations for data analysis, such as algorithms for linear algebra, optimization, and data visualization.

TensorFlow is a library that provides a collection of tools for creating machine learning models, especially those based on neural networks, see → <https://www.tensorflow.org>. The TensorFlow playground allows interactive work with neural networks directly in the browser, see → <https://playground.tensorflow.org>.

PyTorch is a framework for machine learning. Its focus is on tensors and their efficient processing, including using the graphics processing unit (GPU). It can also be used to develop neural networks, including *convolutional neural networks* (CNN).

→ <https://pypi.org/project/torch/>

The required packages must be installed. This can be done using the Anaconda Prompt application in command line mode:

```
(base) C:\Users\Uwe>pip install scikit-learn
```

Other required packages are installed in the same way. Once available, the packages or parts thereof must be imported at the beginning of the program.

The following Jupyter notebook cells show the structure of a decision tree for the weather example already mentioned several times. We start by importing the required packages or components from them. The data is then read from the *csv* file and output once for checking.

```
import pandas as pd
from sklearn import tree
from sklearn import preprocessing

# Reading data
data = pd.read_csv('golfnumeric.csv')
print(type(data))
print(data.head())
```

```
<class
'pandas.core.frame.DataFrame'>
```

	outlook	temperature	humidity	windy	play
0	sunny	85	85	False	no
1	sunny	80	90	True	no
2	overcast	83	86	False	yes
3	rainy	70	96	False	yes
4	rainy	68	80	False	yes

The weather data contains two attributes that are nominal: *outlook* and *windy*. However, the algorithm from the *sklearn* package cannot

process string attributes. Therefore, these must be converted into numbers. These are coded ordinally, as there is a certain order. The coded data is then recombined with the *temperature* and *humidity* attributes and output for checking.

```
# Pre-processing: tree learner does not process
nominal/ordinal
# Columns of numeric values:
X_numerical = data[['temperature', 'humidity']]
# Columns of strings
X_categorical = data[['outlook' , 'windy']]
# Result-column
Y = data['play']

# Coding strings as numbers
nom2num = preprocessing.OrdinalEncoder()
X_ordinal = nom2num.fit_transform(X_categorical)
    # result is an np.Array,
    # that is transformed into a Dataframe:
df_ordinal = pd.DataFrame(X_ordinal, columns=['outlook'
, 'windy'])

X_train = pd . concat([df_ordinal, X_numerical], axis=1,
join="inner")
print(X_train)
```

	outlook	windy	temperature	humidity
0	2.0	0.0	85	85
1	2.0	1.0	80	90
2	0.0	0.0	83	86
3	1.0	0.0	70	96
4	1.0	0.0	68	80

5	1.0	1.0	65	70
6	0.0	1.0	64	65
7	2.0	0.0	72	95
8	2.0	0.0	69	70
9	1.0	0.0	75	80
10	2.0	1.0	75	70
11	0.0	1.0	72	90
12	0.0	0.0	81	75
13	1.0	1.0	71	91

A decision tree is now easy to generate. A *DecisionTreeClassifier* object is created and the tree is constructed using the *Fit* function.

→ Figure 1.27 shows the generated decision tree. In each box, the *value* shows how many data records with *no* or *yes* results were assigned to this node. For example, *value=[0, 4]* characterizes a leaf node in which only the result *play=yes* is assigned.

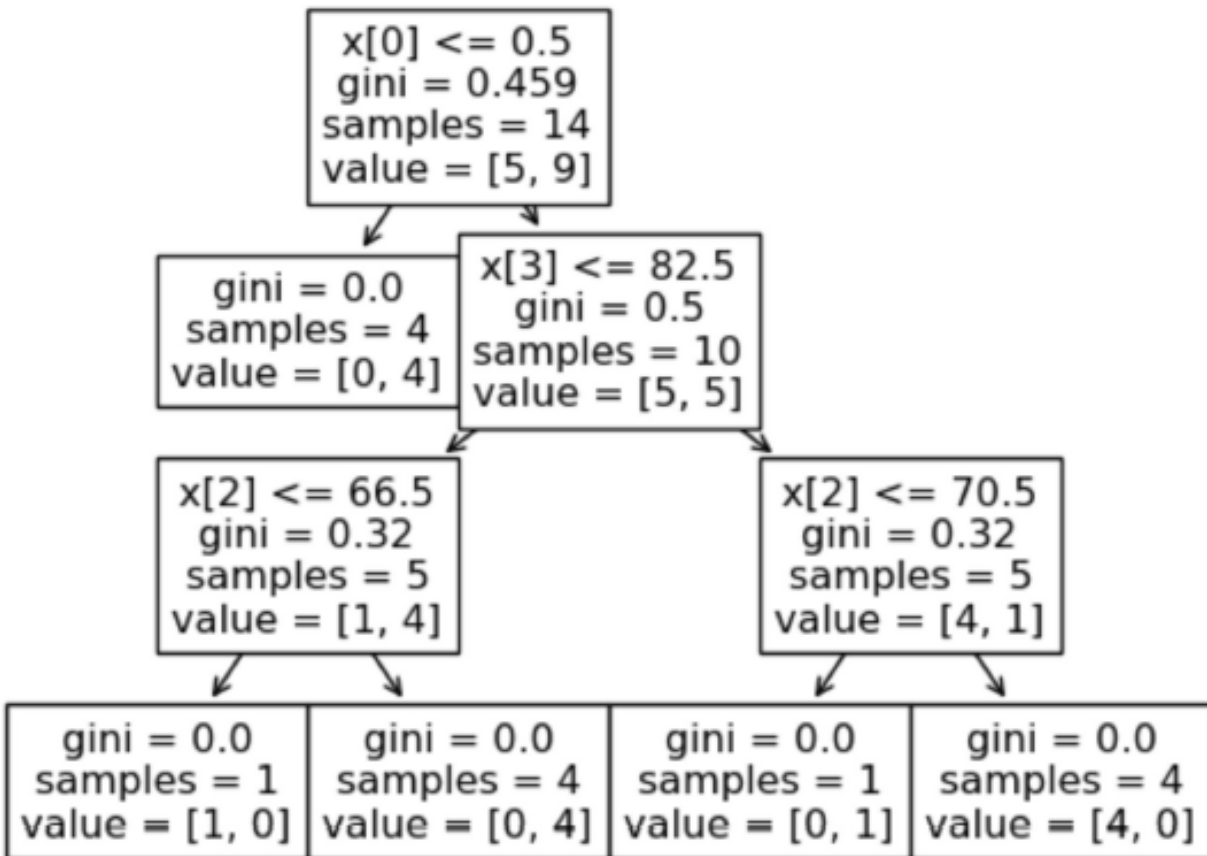


Fig. 1.27 Decision tree for the weather example

```

# Decision tree
dectree = tree.DecisionTreeClassifier()
dectree.fit(X_train, Y)

pred = dectree.predict(X_train)
tree.plot_tree(dectree)

```

In → Section 10.6, the Python Notebook and some of the packages mentioned here are used to develop a neural network for a data mining problem.

Task 1.1

(Data mining tools). Familiarize yourself with the systems presented and implement the examples using the respective software.

2 Fundamentals of data mining

If you file it, you'll know where it is but you'll never need it. If you don't file it, you'll need it but never know where it is.

Tillis' Organizational Principle

2.1 Basic concepts

Data mining aims to extract knowledge from data. But what is data? What is knowledge? How are the terms *data*, *information*, and *knowledge* related?

In our book, we start from a hierarchy of terms that has become established in the literature. The foundation of IT-based processing is *data*.

Definition 2.1

(Data). **Data** is a collection of characters with a structure (syntax).

A distinction is made between

- Unstructured data,
- Semi-structured data, and
- Structured data.

Typical examples of *unstructured* data are images and text. Although web pages consist mainly of text and could therefore be regarded as unstructured data, they do have a structure and are therefore considered *semi-structured* data.

Our book focuses on data mining on *structured* data. Structured data refers to tables in relational databases or data in similarly structured file formats. This also includes spreadsheet file formats and the CSV (comma-separated values) exchange format, which can be edited directly with an editor as ASCII text. The formats mentioned have a fixed structure: the data contained in each data record has a fixed order, and the attributes with their data types are defined.

Let us now return to the hierarchy of terms. If a date, for example the number -4, carries a meaning, then it becomes *information*. If -4 stands for the change in a stock price (in percent), then -4 becomes *information*. Data thus becomes *information* when meaning is assigned to it.

Definition 2.2

(Information). **Information** is data that is linked to a meaning.

Information is therefore the interpretation of data for a specific purpose. Data becomes information when it is viewed in context and given meaning. However, if we have a rule that triggers the sale of a stock, then it is *knowledge*.

Definition 2.3

(Knowledge). Information combined with the ability to use it is referred to as **knowledge**.

Information therefore only becomes *knowledge* when it can be used, or "S understands knowledge K if S uses K whenever appropriate." [→ MN73].

In other sources, the term *information* is placed at the top of this hierarchy. The hierarchy of terms can also be expanded to include the terms *signal* or *sign*, which are then classified below the term *data*. In North's knowledge ladder [→ NK25, S. 39], the terms *action*, *competencies*, and *competitiveness* are placed above the term *knowledge* from a business perspective. Even though data mining can increase a company's competitiveness, we will not consider these terms further here.

Now we can define what we mean by *data mining*.

Definition 2.4

(Data Mining). **Data mining** is the extraction of knowledge from data.

We expand this definition with a few requirements. We want to obtain knowledge that was previously unknown. In other words, we are talking about the extraction of *hidden* knowledge. This knowledge must not be trivial, but must be useful. It must be possible to apply it in a meaningful way. Data mining should also be performed largely automatically. Anyone who has already gained some experience in data mining knows that the last requirement is by the word *largely*. The actual data analysis will usually run automatically, but the preparation requires intensive support from the analyst.

Now that we have defined the term *data mining*, let's turn our attention to the subfields of data mining. Based on the division into unstructured, semi-structured, and structured data, the following subfields of data mining exist:

- Text mining (unstructured data),
- Web mining (semi-structured data), and
- Data mining in the narrow sense (structured data).

Text mining deals with the analysis of texts, i.e., unstructured data. *Web mining* generally works with the semi-structured data already discussed. *Data mining in the narrower sense* works with structured data. Data mining often refers to data mining in the narrower sense. We will limit ourselves to data mining in the narrower sense and only touch on the other two subfields in isolated cases.

We use the terms *data record*, *instance*, *object*, and *pattern* largely synonymously when referring to structured data. This always refers to *one* specific data record in a database table that characterizes an object or instance based on a set of characteristic values. For example, the data record

sunny, hot, high, false, no

can be mathematically considered as a quintuple

(sunny, hot, high, false, no)

and thus as a *single* object. This object is an element of the set of all weather situations; it is an instance of the class of all weather situations.

2.2 Datatypes

After discussing the basic forms in which data can exist (unstructured, semi-structured, structured), we will now focus on structured data and consider the data types that data mining may encounter.

Data can take many different forms. We deal with numbers every day. Numbers can be used for calculations: we can calculate averages, compare numbers, and order them according to their size. Data can also take the form of *small, medium, large, very large*. Like numbers, this data has an order; the data points can be compared according to their size and therefore sorted. However, we cannot perform calculations with this type of data. Finally, there are data types such as the data type *color*. Unfortunately, we have no order here. With the colors red and blue, it is impossible to say which is better, bigger, or more colorful; and calculations are certainly not possible here.

Data types can be categorized as follows based on the two criteria of *order* and *calculation*:

Nominal data is not subject to any ordering. We cannot perform calculations with this data, but only check it for equality: two values are either *equal* or *not equal*.

Ordinal data does have an order relation, such as the relation *less than* ($a < b$), but it is not possible to perform calculations with this data.

Metric data has all the ordering characteristics of real numbers. It is possible to perform calculations with this data.

Metric data types can be further subdivided:

Discrete data types are those that can only be changed incrementally. *Quantities* fall into this category. In programming, these are data types such as *integer*.

Continuous data types are numerical data that can take on any real number value within the definition range.

Strictly speaking, continuous data in computers is also discrete, as it cannot be reproduced exactly due to the limited representation on computers. We will ignore this nuance and assume that such data can be represented with any degree of accuracy.

Continuous, numerical data types can be converted into discrete data, for example by forming intervals. These are then referred to as *discretized* data types.

Even if the values themselves are numerical, the data type does not necessarily have to be metric. Customer numbers, for example, are numbers, but they are nominal data types. If the customer numbers were assigned in chronological order, then the customer numbers are ordinal data. The situation is similar with numerical postal codes. These should not be interpreted as metric values either. The difference between the German postal codes 10117 (Berlin) and 20095 (Hamburg) is greater than the actual distance of 290 km.

Numeric data types are also classified according to the scale on which they are based.

Interval-based data types are characterized by the fact that intervals can be determined, but the zero point has been *arbitrarily* defined. Years are of this type: the Christian calendar defines the year 0 differently than the Unix world, in which 0 corresponds to the year 1970 in the Christian calendar. Similarly, temperature readings in Celsius are interval-based. However, these values can only be used for calculations to a limited extent. For example, 20°C is not (physically speaking) four times warmer than 5°C.

Ratio-based data types have a natural zero point. Distance specifications allow relative comparison alongside addition and subtraction: 60 km is twice as far as 30 km. However, ratio scales depend on the unit of measurement used. 60 means something different when using km as the unit of measurement than when using cm.

Absolute scale-based data types differ from ratio-based data types in that they do not depend on any unit of measurement.

Example 2.1

(Data types). Examples of the data types presented are:

Nominal

- Gender = {female, male}
- Hair color = {blonde, dark blonde, brown, red, black, gray}
- Marital status = {single, married, widowed, divorced}

Ordinal

- School grades = {1, 2, 3, 4, 5}
- Opinion = {agree, partially agree, disagree}
- Verbal categories = {very rich, rich, average, poor}
- Health status = {healthy, sick}

The last example is certainly a borderline case, as we are implicitly assuming here that “healthy” is better than “sick”. If this is not to be taken into account, then it is a nominal data type.

Metric, interval scale

- Years
- Temperature in Celsius or Fahrenheit

Metric, ratio scale

- Measured values such as electricity, wind speed
- Distance
- Body height
- Temperature in Kelvin

Metric, absolute scale

- Years of life
- Number of children
- Number of items

The boundary between ordinal and metric can be quite fluid. Ordinal data is obviously not metric, but we can interpret school grades or earthquake magnitude as metric attributes, for example, by moving away from discrete values and allowing grades such as 1.87.

In particular, ordinal data can be converted into metric data. If we consider an attribute with the values *small*, *medium*, *large*, and *very large*, we can easily convert this into a metric attribute by replacing the values with numbers:

- *small* → 0
- *medium* → 0.3
- *large* → 0.7
- *very large* → 1

We have some freedom when choosing the numerical values. Of course, *small* should be mapped to 0 and *very large* to 1. How we map the other values is up to us. However, the (arbitrary) choice of numbers determines the distance between the values and thus may influence the result.

We can also convert nominal attributes into metric attributes. Binary attributes such as *gender* are transformed into a metric attribute by coding the two values as 0 and 1.

For a nominal attribute with more than two values, a new attribute is introduced for each value. Let's consider the attribute *color*. For each color that occurs in our database, we introduce a separate attribute. A 1 is entered if the color exactly matches the attribute value (e.g., black), otherwise 0. This approach always works, but has the disadvantage that we replace *one* attribute with *several*, thus increasing the size of the data set. This coding is called *binary coding* (→ Section 8.2.5, p. 215). It is often used in neural networks.

Can we convert metric attributes into ordinal ones? This is possible by, for example, forming intervals and then giving them names: *small*, *large*, *very large*.

Such conversions are relevant in data mining because there are methods that work either exclusively or preferably with a specific data type. For example, a continuous, metric attribute is particularly suitable for the *k-nearest neighbor method* (see → Section 5.1). Ordinal and nominal data are not excluded, but they are rather unsuitable. We will discuss the various options for data conversion in detail in → Section 8.2.5.

Which comparison and arithmetic operations can be performed with the different data types? For *nominal* attributes, only a comparison for *equal* or *not equal* is possible. *Ordinal* attributes also allow comparisons for *less than* and *greater than*. *Metric* attributes can also be used for calculations. At a minimum, addition and subtraction are possible, but usually all four basic arithmetic operations. However, multiplication and division are usually not useful for interval-based data types.

Task 2.1

(Data types). Assign the following attributes to the respective data type:

- Postal codes
- ISBN
- Fuel consumption
- Vehicle performance
- House number

Task 2.2

(Data types). Find further examples for all data types.

2.3 Distance and similarity measures

Many applications in data mining are based on comparing data records with each other. When forming clusters (see → Section 3.1 and → Chapter 6), the goal is to group similar objects together. However, this requires that the similarity of two data records can be quantified. This is achieved using so-called distance measures, i.e., measures that quantify the “dissimilarity” of the data records. If a *distance measure* (or *distance function*)

$$\text{dist}(x, y)$$

is given, which measures the distance between two data records x and y , then the similarity

$$\text{simil}(x, y)$$

of the two data records can be defined as a function of this distance measure **dist**. The larger the distance between the data records, the less the similarity:

$$\text{simil}(x, y) = f(\text{dist}(x, y))$$

A distance function must fulfill the following properties:

1. $\text{dist}(x, y) \geq 0$
2. $\text{dist}(x, x) = 0$
3. $\text{dist}(x, y) = \text{dist}(y, x)$
4. $\text{dist}(x, y) \leq \text{dist}(x, z) + \text{dist}(z, y)$

First, of course, the distance between two objects must not be negative. The distance of a data record x from itself must be 0. This requirement can be tightened to

$$\text{dist}(x, y) = 0 \text{ if and only if } x = y.$$

A distance measure must be *commutative*: the distance between x and y is the same as between y and x . The fourth condition (*transitivity*) corresponds to the triangle inequality and requires that the direct distance between x and y is less than (or equal to) the distance resulting from including an intermediate point z .

Let's look at some typical distance functions:

$$\textbf{Hamming distance} \quad \text{dist}_H(x, y) = \text{count}_i(x_i \neq y_i) \quad 2.1$$

$$\textbf{Euclidian distance} \quad \text{dist}_E(x, y) = \sqrt{\sum_i (x_i - y_i)^2} \quad 2.2$$

$$\textbf{Manhattan distance} \quad \text{dist}_{\text{Man}}(x, y) = \sum_i |x_i - y_i| \quad 2.3$$

$$\textbf{Maximum distance} \quad \text{dist}_{\text{Max}}(x, y) = \max_i (|x_i - y_i|) \quad 2.4$$

The Hamming distance counts how many positions the data records differ in. The distance between the data records (*Schulze, 1978, BMW*) and (*Mueller, 1979, BMW*) is 2, as the data records differ in two positions. The Hamming distance does not reflect that the year 1978 differs only minimally from 1979. This is a disadvantage of the Hamming distance. However, one advantage is that it can be applied to any data type, including nominal, ordinal, and metric data.

The Euclidean distance is something we use every day. It is the spatial distance between two locations, or in a generalized or mathematical sense, the distance between two points in n -dimensional space. It can only be applied to metric data.

This also applies to the Manhattan distance. It takes its name from the chessboard-like arrangement of streets in Manhattan. → Figure 2.1 shows that every path from the bottom left to the top right is the same length. The distance is always the sum of the partial distances in both directions.

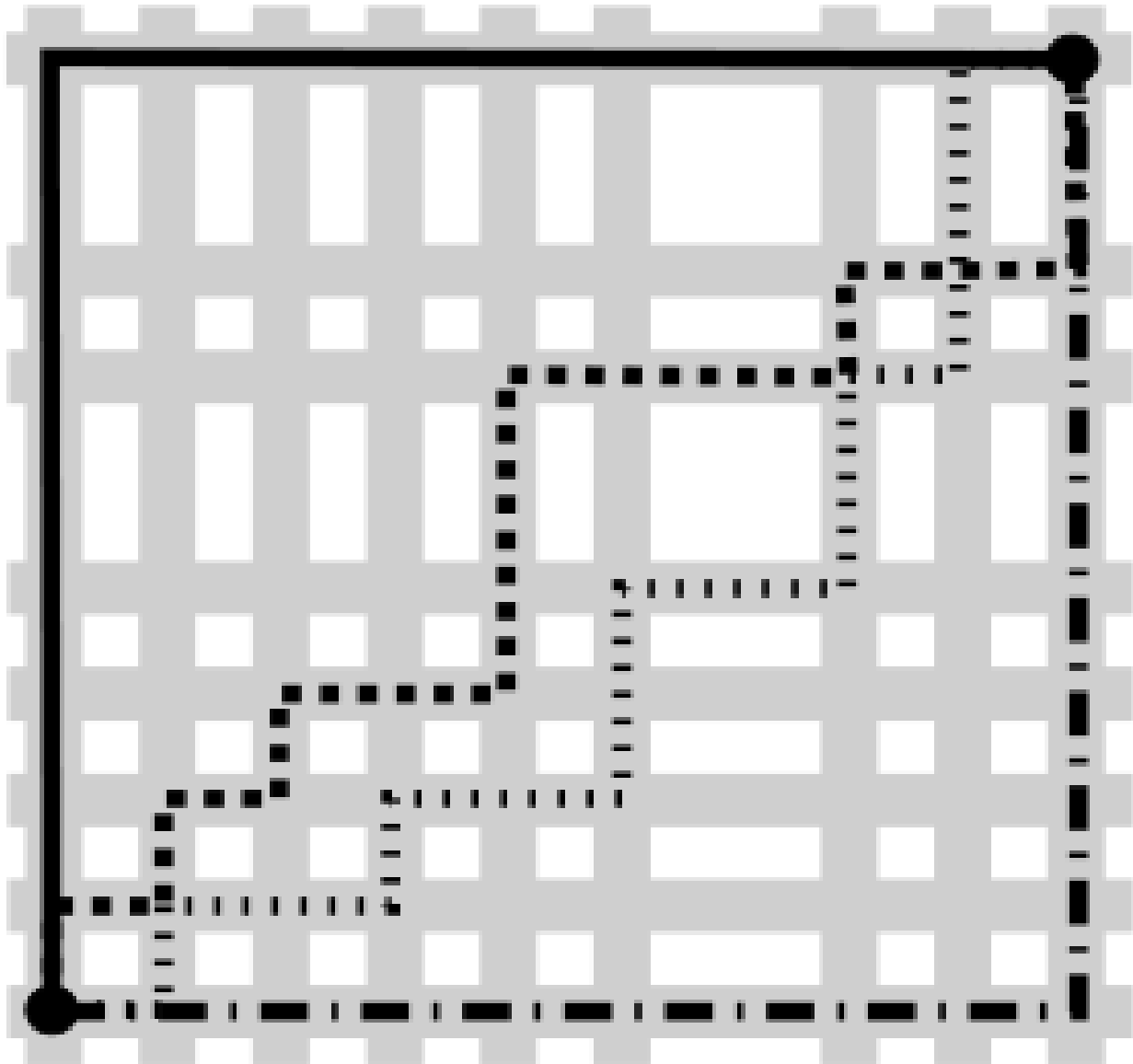


Fig. 2.1 Manhattan distance

The maximum distance also considers the different dimensions separately and selects the largest distance that results across all attributes. It is also known as the Chebyshev distance.

→ Figure 2.2 shows the four different distances for two example points.

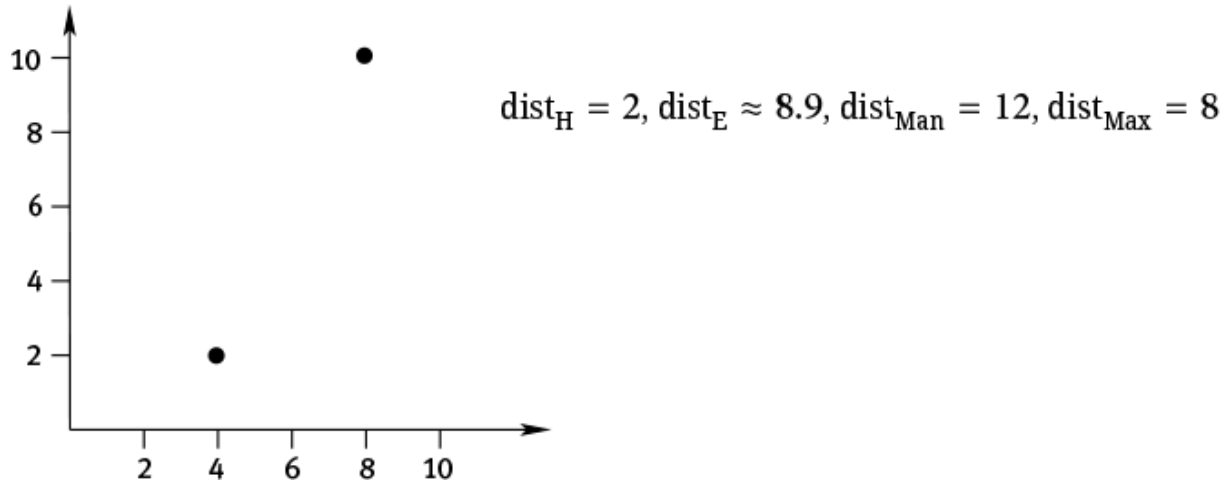


Fig. 2.2 Distances for two example points

There is a generalization of Euclidean distance called *Minkowski distance*:

$$\text{dist}_{\text{Minkowski}}^n(x, y) = \sqrt[n]{\sum_i |x_i - y_i|^n} \quad 2.5$$

For $n = 2$, this gives the Euclidean distance, and for $n = 1$, the Manhattan distance:

$$\text{dist}_{\text{Minkowski}}^1(x, y) = \sqrt[1]{\sum_i |x_i - y_i|^1} = \sum_i |x_i - y_i| \quad 2.6$$

For readers interested in mathematics, it should be noted that for $n \rightarrow \infty$, the Minkowski distance becomes the maximum distance:

$$\lim_{n \rightarrow \infty} \text{dist}_{\text{Minkowski}}^n = \text{dist}_{\text{Max}} \quad 2.7$$

An interesting modification of the Euclidean distance is the *weighted Euclidean distance*:

$$\text{dist}_{\text{EW}}(x, y) = \sqrt{\sum_i w_i \cdot (x_i - y_i)^2} \quad 2.8$$

This distance measure is as well only applicable to numerical attributes. These can be weighted differently by the factors w_i ; the higher the weight, the greater the influence on the distance. Weighted Euclidean distance allows attributes to be given preference, meaning they are assigned a higher priority. The same effect (the preference of certain attributes) can be achieved by appropriate normalization (see → Section 8.2.5).

In the examples, we mostly use the Euclidean distance for numerical attributes. However, you can also experiment with other distance measures; it is not mandatory to use the Euclidean distance for numerical attributes.

So far, we have attributed similarity to distance, i.e., dissimilarity. The smaller the distance, the greater the similarity. There are several *similarity measures* that are not based on distance measures, for example the measures *cosine*, *overlap*, *dice*, and *Jaccard*.

A similarity measure (*similarity function*) **simil** should fulfill the following properties:

1. $\text{simil}(x, y) \geq 0$
2. $\text{simil}(x, y) = \text{simil}(y, x)$
3. $\text{simil}(x, y) \leq \text{simil}(x, x)$

A similarity measure is called normalized if

$$\text{simil}(x, x) = 1$$

The cosine similarity quantifies the similarity of two vectors, thus requiring numerical attributes. It is defined as the cosine of the angle enclosed by the vectors. For vectors pointing in the same direction (i.e., for an angle of 0°), the result is 1. If the vectors are perpendicular to each other (angle 90°), the result is 0. If the vectors point in opposite directions (angle 180°), the result is -1.

$$\text{cosine} \left(x, y \right) = \frac{x \cdot y}{\sqrt{\sum_i x_i^2 \cdot \sum_i y_i^2}} = \frac{\sum_i x_i \cdot y_i}{\sqrt{\sum_i x_i^2 \cdot \sum_i y_i^2}} \quad 2.9$$

A value close to 1 indicates a high degree of similarity between the vectors, while a value close to -1 indicates a high degree of dissimilarity. This similarity measure can be used in the self-organizing maps in → Section 6.7 to determine the winner neuron.

Current tools offer a good selection of distances and even allow you to define your own distance measures. KNIME also has a number of nodes for distance measures.

Task 2.3

(Distance measure and chessboard). The number of steps taken by the king on a chessboard is chosen as the distance. Which distance concept does this correspond to? Which distance measure corresponds to the number of squares passed by the rook?

Task 2.4

(Distance measure). Calculate the pairwise distances between the points (0, 1, 2), (1, 5, 3), and (4, -2, 3). Use all four distance functions listed.

Task 2.5

(Hamming distance). Why is the Hamming distance robust against outliers?

Task 2.6

(Other distance measures). Find other distance measures. For which data types are they suitable? How do they differ from the distance measures discussed here?

Task 2.7

(Combined distance measure). How would you develop a distance measure for data records that are heterogeneous, i.e., that contain mixed data types: nominal, ordinal, and metric?

2.4 Fundamentals of artificial neural networks

An *artificial neural network* is an attempt to create a repository of knowledge that functions similarly to the powerful human brain in order to achieve comparable performance. If one wants to follow this biological model, elements and procedures in a natural neural network must be transformed into the world of computers.

Definition 2.5

(Artificial neural network). An *artificial neural network* consists of a set of artificial neurons, which are linked by directed and weighted connections (compare [LC23]).

→ Figure 2.3 shows a feedforward neural network consisting of an input layer, two hidden layers (also called intermediate layers), and an output layer. A layer consists of a set of individual artificial neurons. The gray shades of the neuron circles indicate different activation states of the neurons.

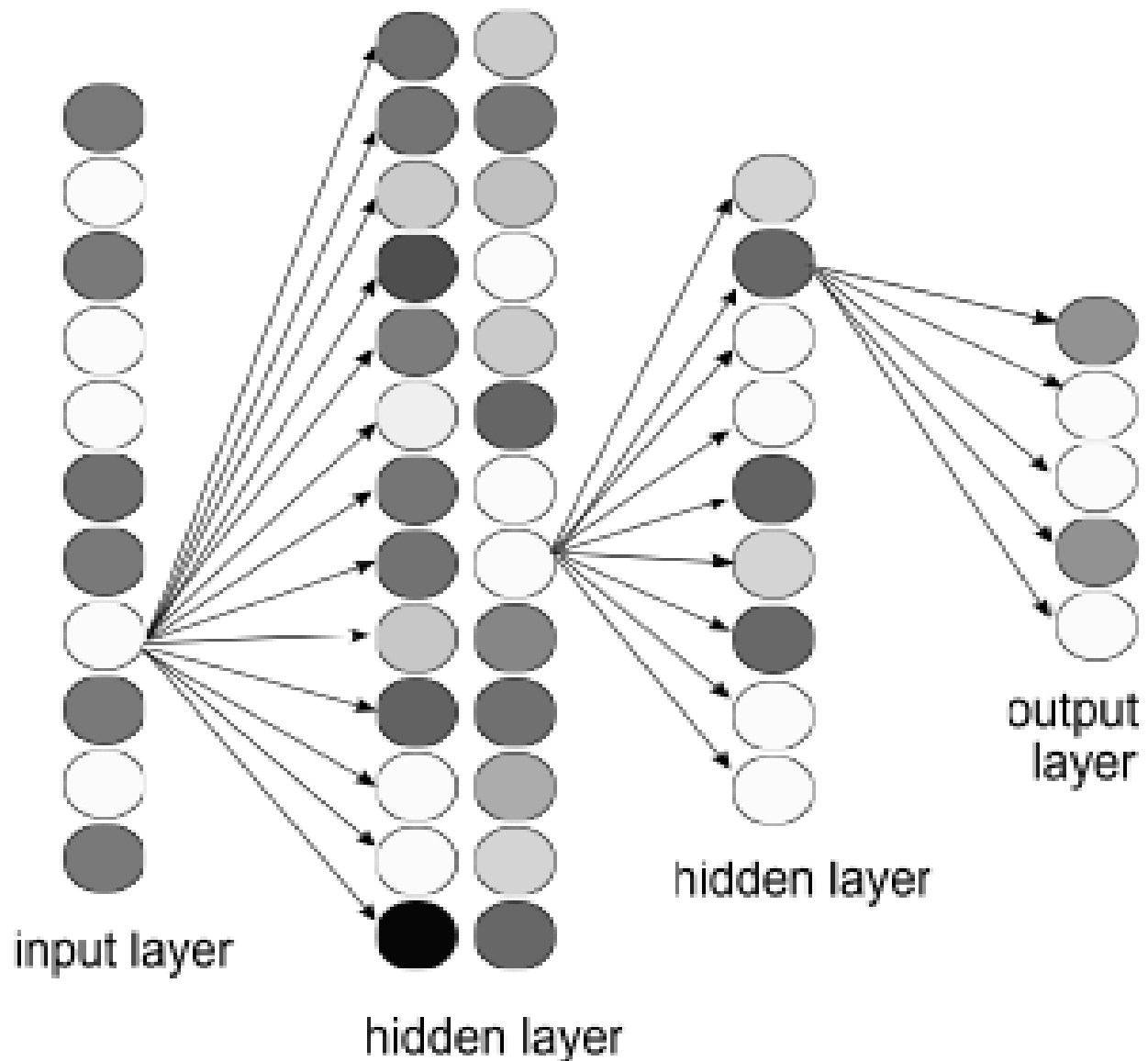


Fig. 2.3 Artificial neural network

Definition 2.6

(An artificial neuron). An *artificial neuron* is a unit that calculates an activation value from the incoming values via weighted connections, taking into account a threshold value and using simple mathematical functions. This value is propagated forward to subsequent neurons.

The *network input* net_η of a neuron η is calculated by adding the outputs of the predecessor neurons o_i , weighted with the corresponding values w_{in} at the connections. An *activation function*, act , uses this value net_η and calculates the activation act_η of the neuron, taking into account a threshold value θ_η . An *output function* with the activation value as a parameter then determines the output o_η of the neuron. → Figure 2.4 illustrates both the structure of an artificial neuron and the calculations that take place in an artificial neuron.

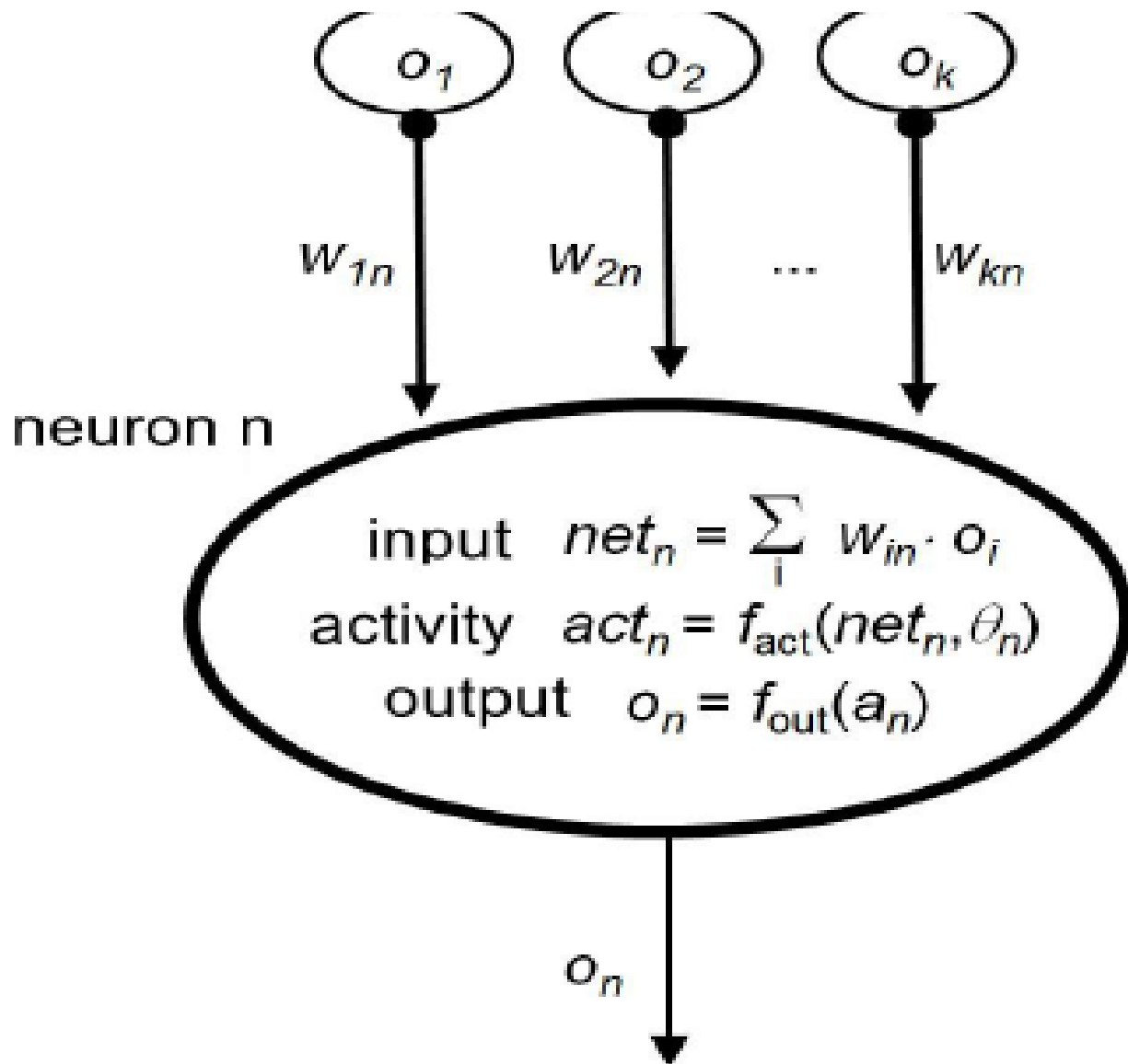


Fig. 2.4 An artificial neuron

Let's take a closer look at the functions used in neurons.

Propagation function

The *propagation function* determines the network input of a neuron. To do this, the outputs of the preceding neurons are multiplied by the corresponding weights at the connections and added up:

$$net_n = \sum_i w_{in} \cdot o_i \quad 2.10$$

Activation function

Various functions with sigmoid behavior can be used as activation functions. Sigmoid behavior refers to a function graph that rises sharply in the vicinity of a point and shows no or only a very

slight increase before and after that point. Sigmoid functions are therefore a kind of switch. In a neuron, they switch the activation of the neuron on or off. Sigmoid functions used in neurons to calculate activation are:

- *Threshold function,*
- *Logistic function,*
- *ReLU function (rectified linear unit),*
- *Hyperbolic tangent.*

The *threshold function* is 0 for all values less than or equal to the threshold $x \leq \theta$ and for all values $x > \theta$, the neuron is activated with the value 1 (→ Figure 2.5). The *threshold function* can be used for the delta learning method in feedforward neural networks without any hidden layer. Feedforward neural networks without any hidden layer are called *perceptrons*.

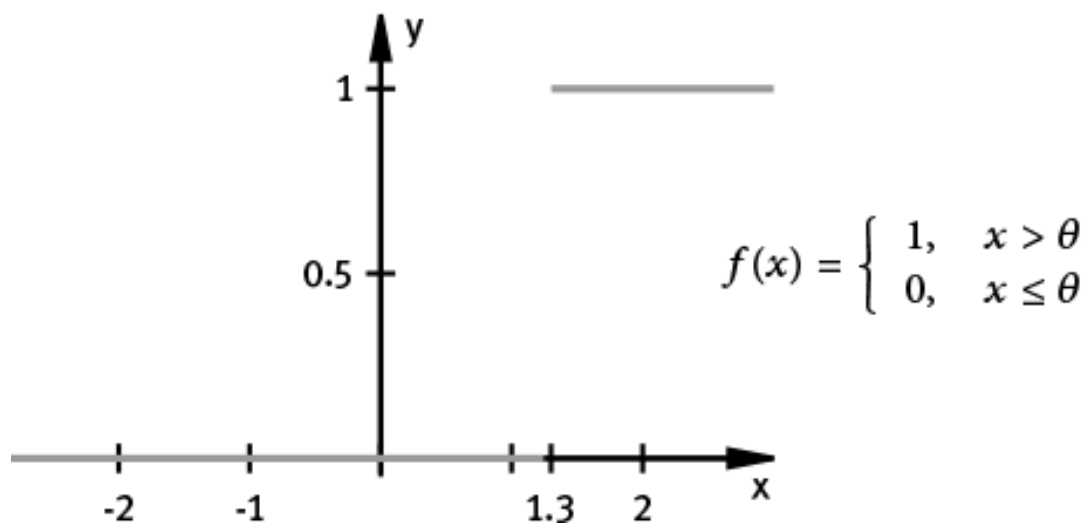


Fig. 2.5 Threshold function for $\theta = 1.3$

A threshold function is not continuously differentiable and therefore cannot be used for learning methods based on a gradient descent method.

The *logistic function* (→ Figure 2.6) is continuously differentiable and is the activation function frequently used in smaller feedforward neural networks.

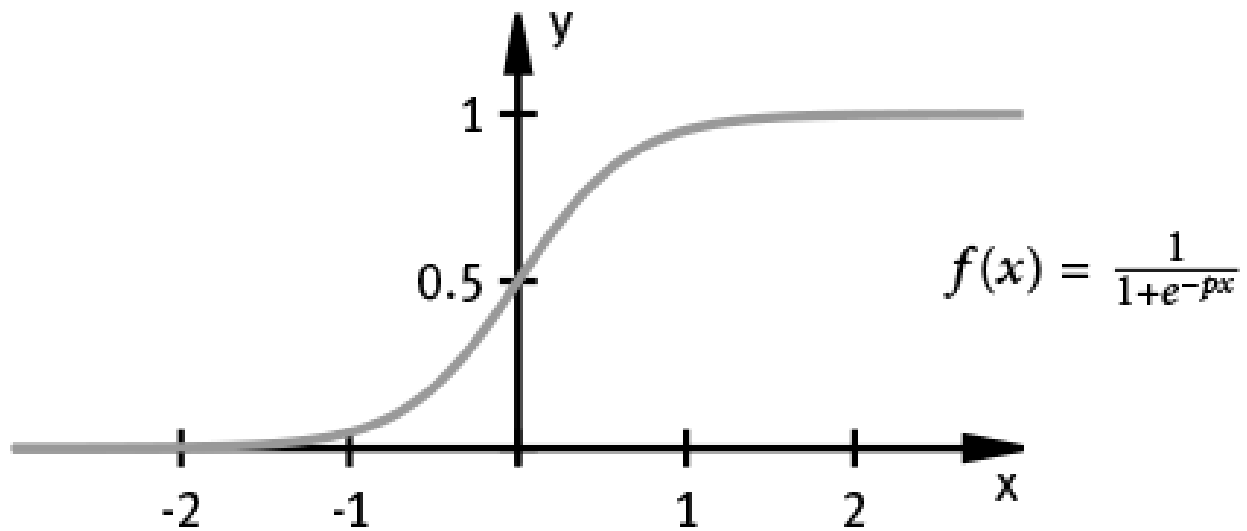


Fig. 2.6 Logistic function

The logistic function is easily differentiable and can be used in learning methods based on a gradient descent method. The parameter p can be used to control the steepness of the curve in the switching point range, which in a neuron is the threshold value. The first derivative of the logistic function used to calculate the change in weight is:

$$f'_{log}(x) = f_{log}(x) \cdot (1 - f_{log}(x)) \quad 2.11$$

The ReLU function $f(x) = \max(0, x)$ (→ Figure 2.7) achieves good results, especially in networks with many hidden layers.

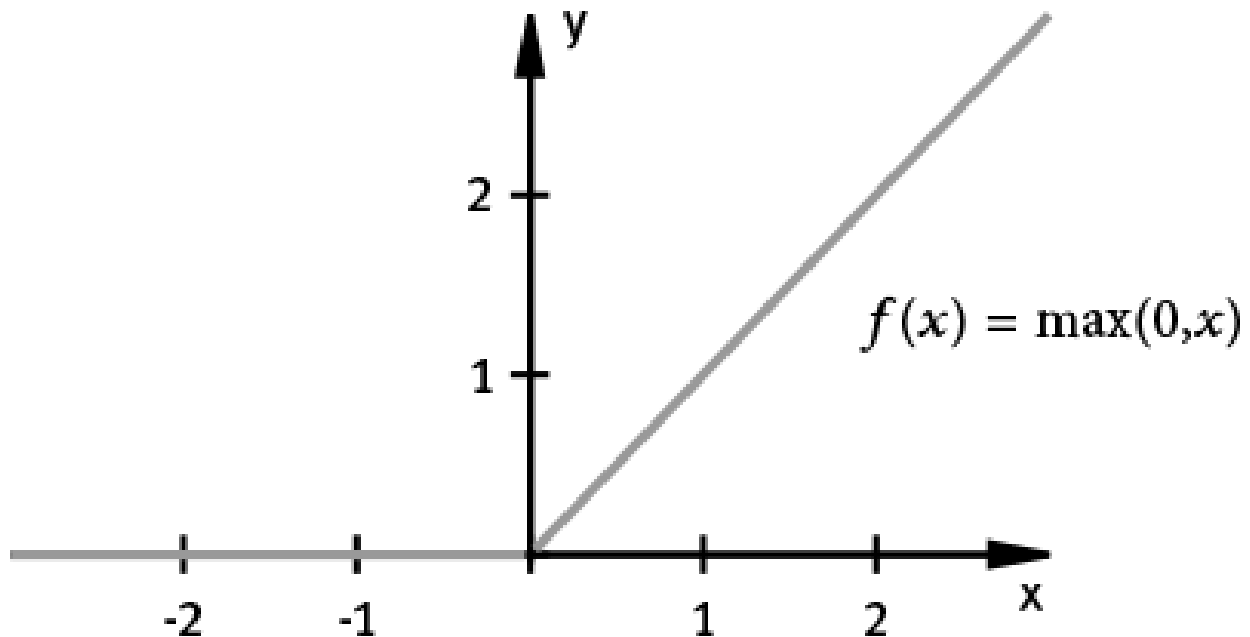


Fig. 2.7 The ReLU function

The *hyperbolic tangent* is used when the range $[-1, +1]$ is considered for the activation of the neuron. Like the logistic function, the *hyperbolic tangent* (see → Figure 2.8) is differentiable and is used in particular in the learning processes of multilayer feedforward neural networks. The first derivative used to calculate the weight change is:

$$\tanh'(x) = 1 - \tanh^2(x)$$

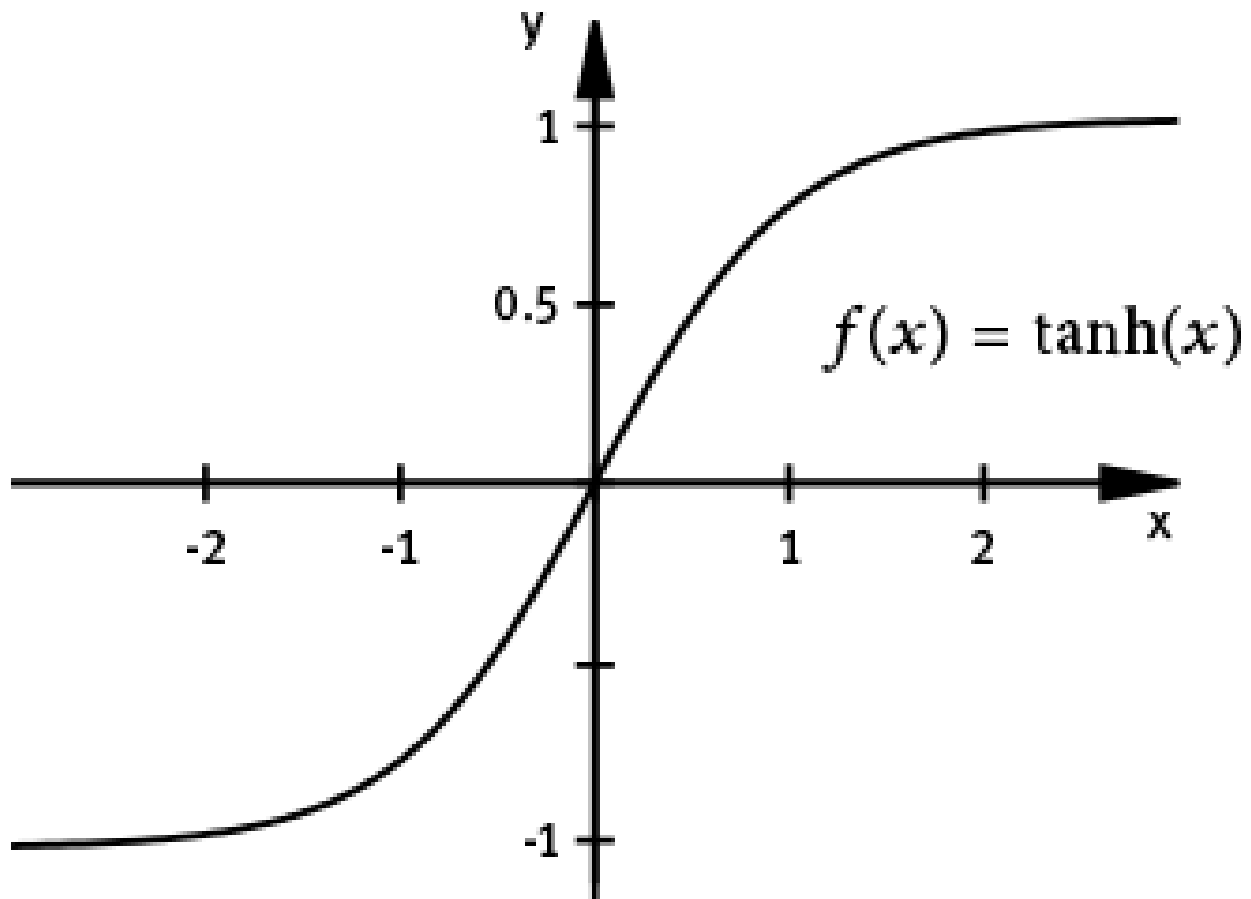


Fig. 2.8 The hyperbolic tangent function

Output function

In many cases, the *identity function* $f(x) = x$ is used as the *output function*. This means that the activity of a neuron is also the output value of the neuron: $o_n = act_n$. In data mining applications, a threshold function can be used in the output layer, which then generates a binary output, 0 or 1. Instead of the usual rounding function, a self-defined threshold, for example less than 0.5, can be used to significantly improve the result.

Example neuron

Let's consider a simple "network" consisting of only one working neuron that simulates the logical AND function (see → Figure 2.9).

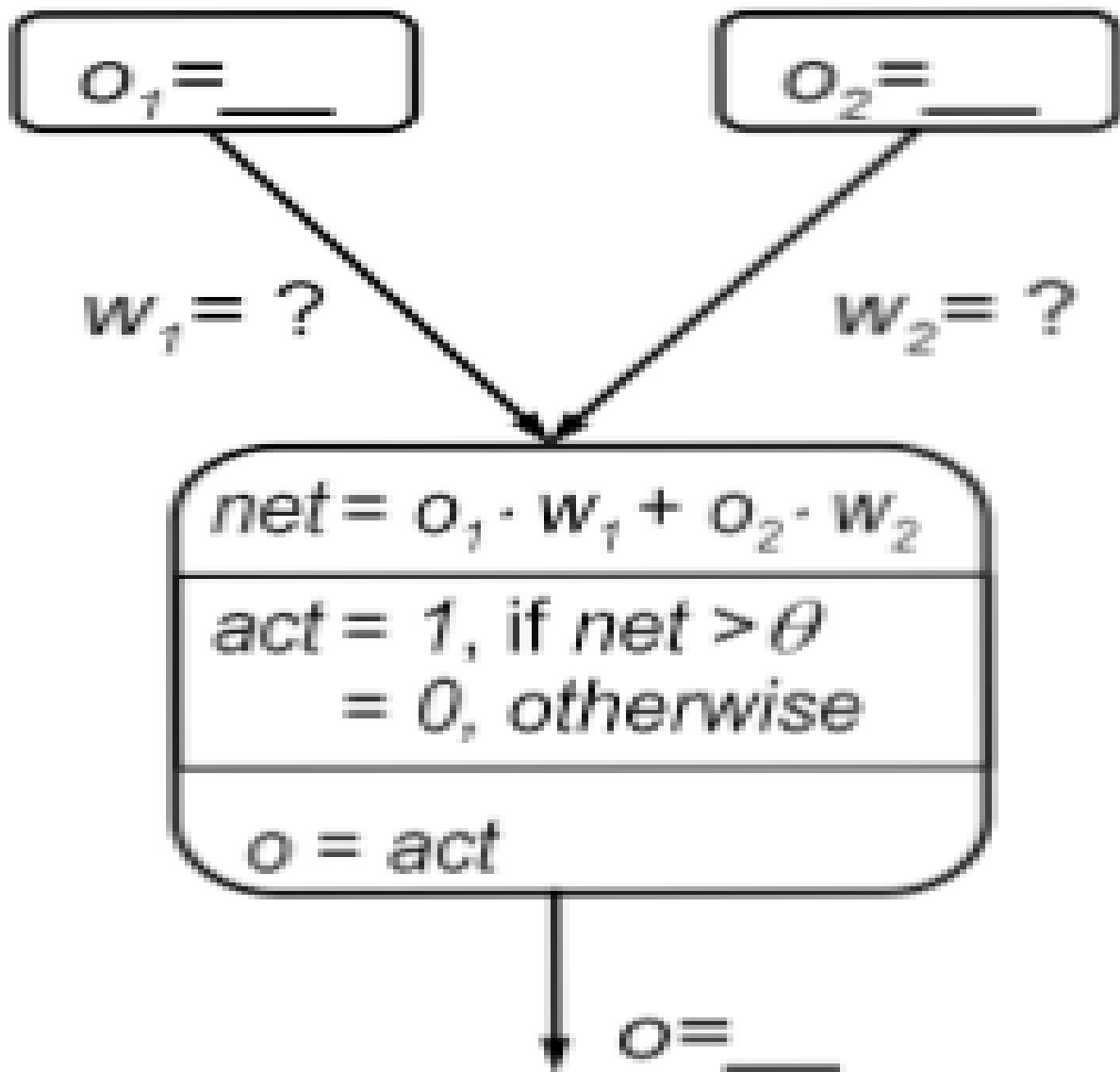


Fig. 2.9 A neuron acting as an AND switch

The input layer consists of two neurons that provide the network input. The activations are set externally. These activations correspond to the values of the input patterns or input data. If both inputs are 1 (i.e., *true*), the network output should also be 1, otherwise 0. This behavior must be achieved by the single neuron of the second layer. The neuron is thus an output neuron.

In typical neural network applications, the weights of the connections and the threshold values of the neurons are learned through training. In this example, we will determine them manually. The question is therefore: What values can we use for w_1 , w_2 , and θ so that the neuron exhibits the desired behavior? We assume that the output function is the identical function $o = act$. The answer is that there are infinitely many possibilities. One is $w_1 = w_2 = 1$ and $\theta = 1.2$.

The processes and calculations in a neuron presented in this section are used in all artificial neural networks in this or a similar form.

2.5 Logic

We cannot provide an introduction to *logic* here. However, since many formal methods of knowledge representation are based on logic and data mining claims to extract knowledge from mass data, we rely on a basic understanding of logic. To this end, we define some terms from mathematical logic here.

Logic, or more specifically *logical formulas*, are used to represent knowledge. But what is a logical formula? We answer this question within the framework of first-order predicate calculus (PC1), which was introduced at the end of the 19th century by the German mathematician and philosopher Gottlob Frege in his *Begriffsschrift* [→ Fre79]. A detailed introduction to this formal logic can be found in [LC23], among other sources.

A *logical formula* is structured according to the following rules:

Definition 2.7

(Logical formula).

1. If p is an n -ary¹ predicate symbol and $t_1, \dots, t_n$ are terms, then $p(t_1, \dots, t_n)$ is also an (*atomic*) logical formula.
2. If A and B are logical formulas, then the following operations are also logical formulas:
 - *Conjunction*: A and B ($A \wedge B$),
 - *Disjunction*: A or B ($A \vee B$),
 - *Negation*: not A ($\neg A$),
 - *Implication*: If A then B ($A \rightarrow B$), and
 - *Equivalence*: A if and only if B ($A \leftrightarrow B$).
3. If x is a variable and F is a logical formula, then the following so-called *quantifications* are also logical formulas:
 - *Universal quantification*: For all x F applies ($\forall x F$),
 - *Existence quantification*: There exists an x , for which F applies ($\exists x F$).
4. There are no other logical formulas.

The concept of a *term* is used in the building rule for logical formulas.

Definition 2.8

(Term).

1. Every *constant* is a term.
2. Every *variable* is a term.
3. If f is an n -ary function and $t_1, \dots, t_n$ are terms, then $f(t_1, \dots, t_n)$ is also a term.
4. There are no other terms.

The effect of logical connectives is shown in the form of a truth table (→ Table 2.1).

Tab. 2.1 Table of truth values for logical operations

A	B	$A \vee B$	$A \wedge B$	$\neg A$	$A \rightarrow B$	$A \leftrightarrow B$
T	T	T	T	F	T	T
T	F	T	F	F	F	F
F	T	T	F	T	T	F
F	F	F	F	T	T	T

Based on the truth values, consider that the implication $A \rightarrow B$ is equivalent to the disjunction $\neg A \vee B$. If A is false, the implication $A \rightarrow B$ (IF A THEN B) is always true. Or B applies, i.e., if A is not false, then B *must* also apply.

Predicate symbols are used to formally express properties of objects or relationships between objects using logical formulas. Here are a few examples:

```
female(anna)
mortal(socrates)
green(house1)
goodCustomer(meier)
child(anna, paul, beate)
class(meier, goodCustomer)
distance(a, b, 100)
```

Not all logical formulas can really be considered knowledge. A logical formula such as `goodCustomer(x)` does not say much:

- Is there a x that is a good customer?
- Is there no x ?
- Are all x good customers?

If x is now quantified and all variables in a logical formula are generally quantified, a truth value can be assigned to this formula:

$$\forall x \text{ goodCustomer}(x) \rightarrow \text{discount}(x)$$

These special logical formulas (the *statements*) are of interest for the representation of knowledge. Statements are logical formulas to which a truth value, *true* or *false*, can be assigned. According to our definition of knowledge as information combined with the ability to apply it, knowledge is always correct. If knowledge is represented by logical formulas, these are obviously only formulas to which we assign the truth value *true*. The statement “If x is a good customer, then customer x will be granted a discount” is such a logical formula, which is to be understood as a guideline for action and thus regarded as true.

If-then statements are used very frequently in this book; see $\rightarrow$ Section 3.2 or $\rightarrow$ Chapter 4. The following statement is taken from $\rightarrow$ Figure 3.2:

IF age = 31...40 AND income=high THEN creditworthiness = very good

These sentences are called *rules*. They are special logical formulas in the sense of predicate logic.

Definition 2.9

(Rule). A *rule* is a logical formula with the following properties:

1. The structure is:

$$\forall x_1, x_2, \dots, x_n (A_1 \wedge A_2 \wedge \dots \wedge A_K) \rightarrow (B_1 \wedge B_2 \wedge \dots \wedge B_l)$$

2. No other variables than $x_1, x_2, \dots, x_n$ appear in A_i or B_j . Thus, all variables in the formula are universally quantified.
3. The A_i and B_j are atomic formulas or negated atomic formulas.

The if-then statement above corresponds to the requirements for a rule and can be written as a logical formula as follows:

$$\forall x (\text{age}(x) \geq 31 \wedge \text{age}(x) \leq 40 \wedge \text{income}(x, \text{high})) \rightarrow \text{creditworthiness}(x, \text{veryGood})$$

Since a rule is a logical formula in which all variables are universally quantified, quantification is not shown in practice. In addition, for reasons of readability, the IF-THEN notation is used instead of the implication sign ($\rightarrow$):

IF $\text{age}(x) \geq 31$ *AND* $\text{age}(x) \leq 40$ *AND* $\text{income}(x, \text{high})$ *THEN* $\text{creditworthiness}(x, \text{ver$

One goal of data mining is to develop a set of rules that describe a classification or a prediction. Once such a set of rules has been developed, the goal of extracting knowledge from mass data has been achieved: the knowledge is now available in the form of rules.

It can be said that the knowledge is represented as a set of logical formulas. Logic is therefore the basis of this representation of knowledge.

Rules can be represented as logical formulas, as described here, but they can also be specified in the form of a table ($\rightarrow$ Section 4.1) or contained in a decision tree ($\rightarrow$ Section 4.2). In a decision table, each column corresponds to a rule; in a decision tree, each path from the root node to a leaf represents a rule.

2.6 Unsupervised and supervised learning

Two *learning strategies* can be distinguished for training: unsupervised learning and supervised learning.

Unsupervised learning

The patterns to be discovered are completely unknown. There are also no examples that specify a grouping or classification. An example of unsupervised learning is cluster analysis ($\rightarrow$ Section 3.1 and $\rightarrow$ Chapter 6). The task is to find groups of similar objects and combine them into subsets, i.e., clusters. No solution is specified here, so we cannot compare the solution developed by an algorithm with a desired cluster set.

Supervised learning

Examples are provided for which the result is given. For example, if we have data records for good and poor customers that contain customer characteristics on the one hand and the classification into good and poor customers on the other, then we know which customer belongs to which

category. Now we can check whether our methods really deliver a correct class assignment (see Classification, → Section 3.2 and → Chapter 5).

Data mining begins with a given set of examples which is called the *instance set* or *example set* E . This set E is used, for example, to form clusters. When forming clusters, we create subsets that hopefully represent a good division of the given data. In this task, we do not know in advance what the optimal clusters will look like. In this case, we can only learn in an *unsupervised* manner.

If, on the other hand, we have a set E with an existing classification (good/poor customer) and want to generate a model for classifying new data, we have the option of checking whether the model works well *before* using it. We do not use the entire set E for *learning*, but only a subset of E : the so-called *training set* $T \subset E$. Another subset of E (usually $E \setminus T$) is used to *test* or *validate* what has been learned. This set is called the *test set* (or *validation set*) $V \subset E$. We thus apply our learned model to V and check whether our prediction matches the actual class. This process is called *supervised learning*. It is explained in detail in → Section 9.4.

The *supervised* learning process is illustrated in → Figure 2.10.

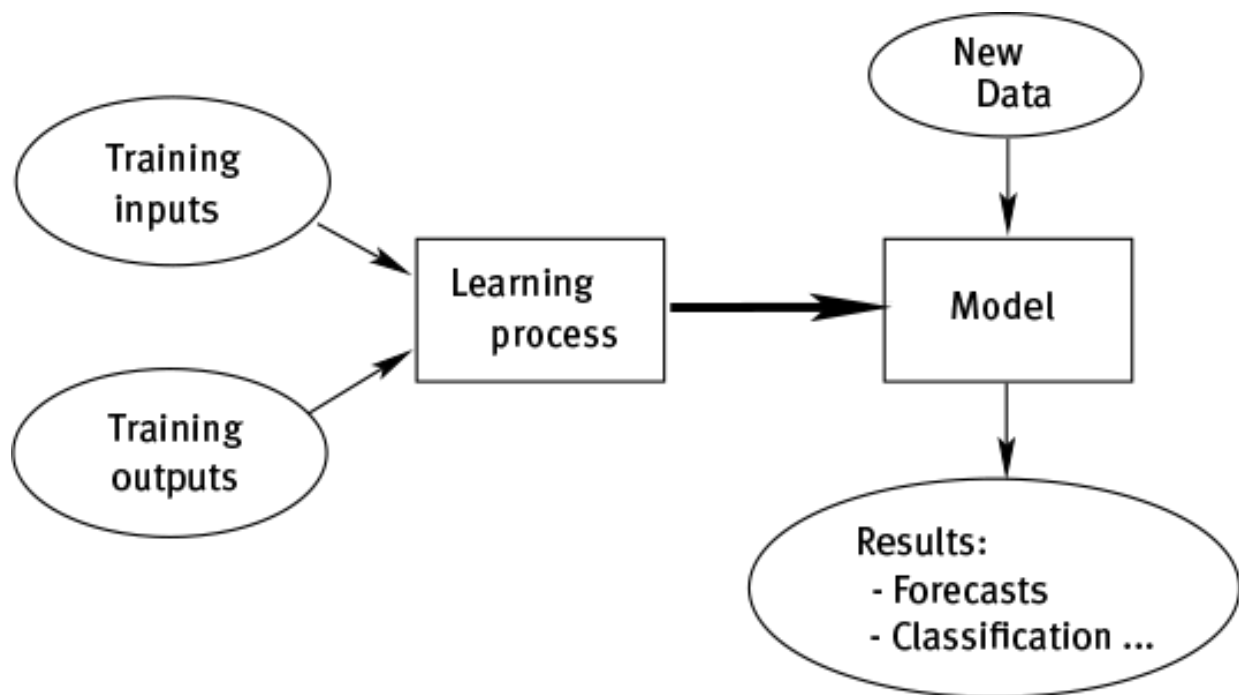


Fig. 2.10 Creating and applying a model

First, a model is generated based on the given training examples and the corresponding training outputs. This model is then applied to new data.

In supervised learning in particular, it is striking that we tacitly assume something, namely that the given sample data is representative of the problem, so that the problem behavior will not change in the future:

The behavior learned from past data is also applicable to future situations.

Every method that attempts to predict future events based on past data is based on this assumption. While this assumption may not always be justified, if we do not accept it as valid, we

should not even begin with data mining (see the comments on data quality in → Section 8.1).

As already mentioned in → Section 1.4, there is another learning strategy known as reinforcement learning. While in supervised learning a model is first generated and then applied, in reinforcement learning a model is developed “on the fly”: first, a model is generated in a *supervised* manner, and then this model is applied and simultaneously further *developed* using new application examples. Reinforcement learning is not discussed further in this book.

3 Application classes

All great discoveries are made by mistake.

Young's Law

Data mining is used to pursue a variety of goals. A typical situation is one in which a large amount of data is available and it is unclear what can or should be searched for. The first step is then to form groups of similar objects. This is referred to as *clustering* (→ Section 3.1). Once such clusters have been found, they are given names. In other words, classes are formed. If the data is already classified (for example, customer data classified as poor, good, or very good customers) then techniques are needed that can predict whether new customers will be poor, good, or very good customers. This problem is addressed by *classification* methods (→ Section 3.2). Similar to classification is the *prediction of numerical values* (→ Section 3.3).

It is also possible to query relationships between the attributes in our database:
People who buy feta cheese also buy olive oil.

The focus here is not on predicting a specific target attribute based on the values of other attributes, but rather on establishing correlations between arbitrary attributes. This is referred to as *association analysis* (→ Section 3.4).

Texts or web pages can also be analyzed using data mining methods. We will briefly discuss this in → Sections 3.5 and → 3.6.

We will not consider other application classes, such as time series analysis.

The application classes can be divided into two categories. *Clustering* belongs to the *descriptive* methods. These methods create models that describe or explain something. In clustering, the data set is divided into subsets in such a way that the objects in each subset are similar to each other.

The second category creates *predictions*, which include *classification* and *numerical forecasts*. Association analysis generates *descriptive models*,

but it can also be used for predictions. The same applies to web mining and text mining: both can deliver predictions as well as descriptive models.

In addition to the application classes listed, there are other fields of application. *Process mining*, as the name suggests, deals with the analysis of processes. Process mining has now become a separate field of study, so we will only outline this area here. Detailed introductions are available in the relevant literature, for example [→ Aal16].

There are three types of process mining:

Discovery Processes are to be extracted from log data. The goal is to identify processes and create process models.

Conformance The extent to which real processes comply with the specifications of an existing process model is checked. Do our processes really run as specified?

Enhancement The goal is to enhance existing process models.

Data mining techniques are used in all of these applications. The basis of every process mining technique is event data, which is usually available in the form of log files.

3.1 Cluster analysis

The goal of cluster analysis (*clustering* or cluster formation for short) is to divide a given set E ($E \subseteq X$) into subsets (*clusters*). The objects within a cluster should be as *similar* as possible to each other, whereas objects from different clusters should be as *dissimilar* as possible. In addition to the set of instances, a cluster analysis requires a *distance* or *separation function* (see → Section 2.3) in order to quantify the similarity of the objects.

Since different cluster formations are possible, a *quality function* (see → Chapter 9) is also required, which allows cluster formations to be compared.

Given are

- an instance space X ,
- an instance set $E \subseteq X$, also called example set,
- a distance function $\text{dist}: X \times X \rightarrow \mathbb{R}^+$

- a quality function $quality : 2^{2^X} \rightarrow \mathbb{R}$

We are looking for a *cluster set* $C = \{C_1, \dots, C_k\}$ with the following properties:

- $C_i \subseteq E$
- $quality(C) \rightarrow \max$
- $C_i \cap C_j = \emptyset$ (if $i \neq j$)
- $C_1 \cup \dots \cup C_k = E$

The last two requirements state that

- no object is assigned to multiple clusters and
- each object belongs to one cluster.

As we will see, both requirements can be dropped. For example, in the case of outliers, it makes sense to leave them isolated, i.e., outside of all clusters.

The goal of a cluster analysis can be formulated with the help of a distance function *dist* in such a way that the given objects are divided into subsets so that the distance between the objects within a cluster is smaller than the distance to the objects in other clusters:

$$\forall C_i, C_j \in C (i \neq j) : \forall x_k, x_l \in C_i, x_m \in C_j : \text{dist}(x_k, x_l) < \text{dist}(x_k, x_m)$$

This requirement is usually too strict, because → Figure 3.1 already shows that an object at the edge of a cluster may well be closer to an object in another cluster than to some objects in the same cluster. We will therefore require that objects be assigned to the cluster whose *representative* they are closest to.

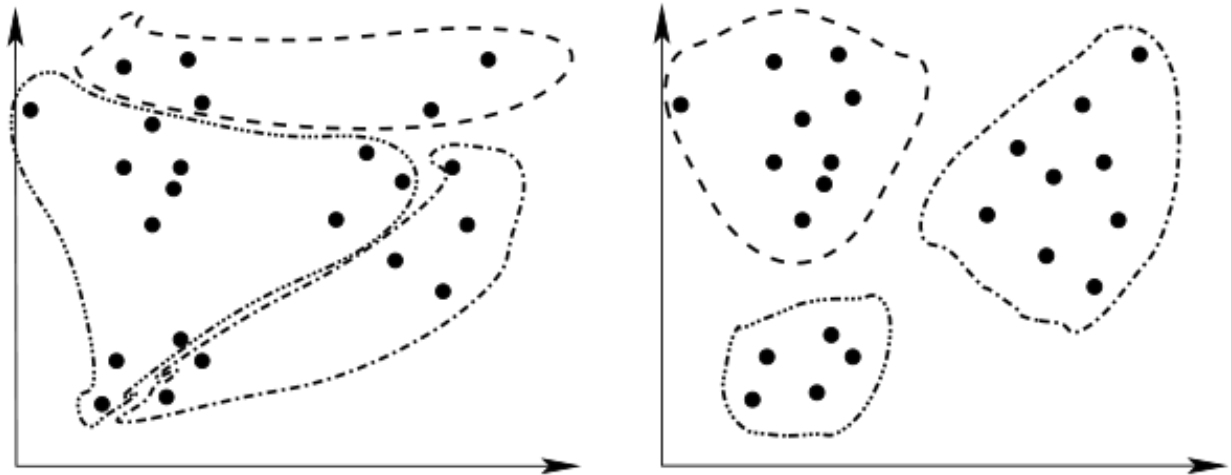


Fig. 3.1 Poor and good clustering

The *quality function* describes how good a cluster formation is:

$$\text{quality}(C = \{C_1 \subseteq E, \dots, C_k \subseteq E\}) \rightarrow \mathfrak{R}$$

In most cases, the quality function is based on the distance function *dist* or on a similarity function *simil* (see → Section 2.3 and → Chapter 9). → Figure 3.1 shows two different cluster formations. The cluster formation on the right part of the figure is obviously better than the one on the left. → Section 9.5 discusses how a quality function can also recognize what we humans see immediately.

Why do we need clusters? What characterizes successful applications of cluster analysis? For example, we can use cluster analysis to identify *homogeneous customer groups* in order to create offers tailored to each customer type. Clustering is also used in the field of optical character recognition (OCR). In this case, clustering techniques are used to search for similar letters and combine them into letter groups in order to develop specialized classifiers for character recognition in the next step.

Which clustering methods do we cover?

- The classic **k-means algorithm** is covered in → Section 6.2.
- In → Section 6.10, we consider a fuzzy variant (**fuzzy c-means**).
- In → Section 6.3, we discuss both the **k-medoid method** and two special variants of this method class: **PAM** and **CLARANS**.

- The **expectation maximization** approach is described in → Section 6.4.
- Some variants of **agglomerative clustering** as a variant of hierarchical clustering are presented in → Section 6.5.
- In → Section 6.6, we discuss **density-based clustering**, in particular **DBSCAN**.
- Some architectures of **artificial neural networks** can be used for clustering:
 - **Self-organizing maps** (→ Section 6.7),
 - **Neural gases** (→ Section 6.8), and
 - **Adaptive resonance theory** (ART networks) (→ Section 6.9).

Cluster analysis is often referred to as *class formation*, as it searches for previously unknown classes of similar objects. Cluster formation is part of *unsupervised learning*.

3.2 Classification

The aim of *classification* is to assign new data records to existing *classes*. For example, customers can be assigned to the class of customers with normal credit rating or the class of those with very good credit rating based on their data. A model is developed using a specified training set of customer data, data sets for which the classification feature of *credit rating* is known.

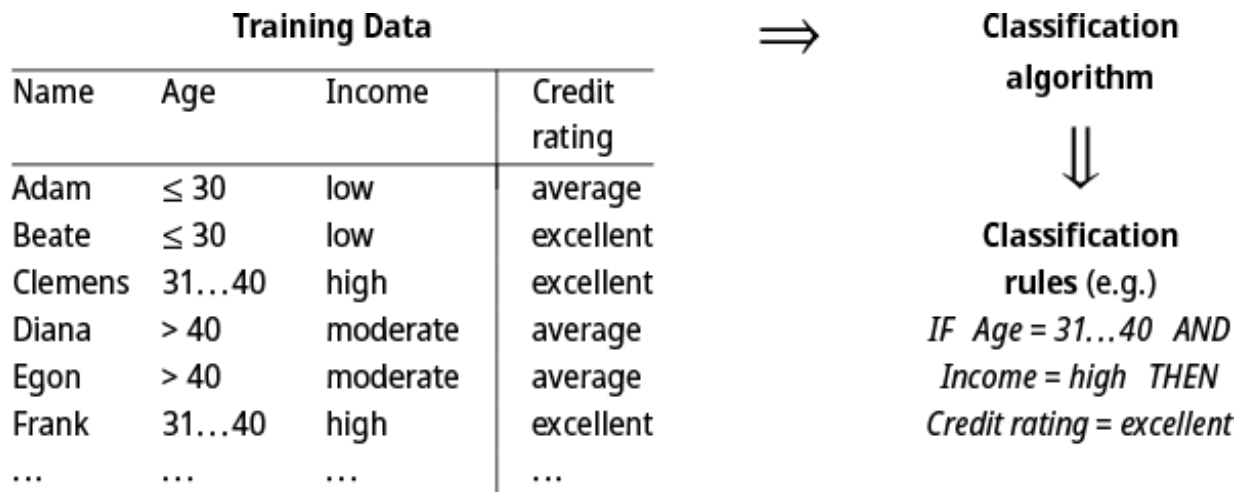


Fig. 3.2 Classification, training

This model is then tested on test data (→ Table 3.1) and corrected if necessary until it has only a low, acceptable error rate for the test data. In order to determine this error, the classification for the test data must also be known for the test set. For this learning process, both the classes and the affiliation of the training and test objects to one of these classes are therefore known in advance. This is referred to as *supervised learning*.

Tab. 3.1 Classification, test phase

Test samples				
Name	Age	Income	Credit rating	Rating by rules
Gerda	31...40	high	excellent	excellent
Hanno	31...40	high	<i>average</i>	<i>excellent</i>
Inge	>40	high	excellent	...
...	...	...	...	...

If the classification model learned is good enough, it can then also classify new data records for which no class membership is yet known, see → Table 3.2. This is the *application phase*.

Tab. 3.2 Classification, application phase

New data				
Name	Age	Income	Credit rating	Rating by rules
Joe	31...40	high	?	excellent
Karl	...	...	...	...
...	...	...	...	...

The role of the learning process is to recognize how the given characteristics (age, income) influence the classification (credit rating).

A clear classification result, i.e., a clear class assignment, is not always necessary or desirable. Often, the *probability* of belonging to a class is also of interest. An object may then belong to several classes, with different probabilities.

Classification is probably the most widely used application. There are numerous examples that predict future customer behavior based on historical data. Will the customer respond to an advertising campaign, or can shipping costs be saved? Can credit rating be predicted? Character recognition, as mentioned above, is also a form of classification: Is the character in question an A?

Which classification methods are covered in this book?

- The **k-nearest neighbor method** (kNN), an instance-based method, is covered in → Section 5.1.
- The generation of **decision trees**, in particular the ID3 and C4.5 algorithms, is presented in → Section 5.2.
- A probability-based method, the **naive Bayes algorithm**, is discussed in → Section 5.3.
- Artificial neural networks, **feedforward neural networks**, are also suitable for classification (→ Section 5.4).
- The basic principle of **support vector machines** is presented in → Section 5.5.

Association analysis methods (→ Section 3.4) can also be used for classification. The association rules are then restricted in such a way that the target, i.e., the classification feature, always appears on the right-

hand side of the rules. These rules are then referred to as classification rules (see → Chapter 7, → Section 4.3 and page 189).

3.3 Numerical prediction

While classification predicts the appropriate class for an object or data record, *numerical prediction* aims to forecast any numerical value. Numerical prediction is usually done by *approximating* a function using sample data. The values of future data sets are calculated on the basis of training data consisting of data records and the associated function values. The available data is used to calculate a function that approximates the actual course of the data. This function can then be used to calculate a corresponding predicted value for new data.

From a given function

$$y = f(x)$$

only a set of instances E is known, which consists of given data records

$$X = \{x_1, \dots, x_n\}$$

and associated target values

$$Y = \{y_1, \dots, y_n\}$$

We are now looking for a function

$$y' = h(x)$$

that reflects the relationships between the instances and target values, i.e., the unknown function f , as accurately as possible. The error between the calculated and actual values should be minimized.

Strictly speaking, classification and numerical prediction fall into *one* category; both aim to predict something. The difference between the two is that classification predicts discrete values, such as *good* or *poor* customer. So there are only a finite number of values (in practice rather few) that the target attribute can take on. In numerical prediction, on the

other hand, numbers are predicted that come from a large (infinite) numerical range of values.

In practice, we often encounter numerical prediction. Examples include the prediction of sales figures for inventory optimization, temperature forecasts, and stock price predictions.

Methods for numerical prediction include the following:

- **Linear regression**
- **Regression trees**
- **Feedforward neural network** (→ Section 5.4)
- **K-nearest neighbor** (→ Section 5.1, real-valued functions)

We do not cover numerical prediction in a separate chapter in this book, but instead discuss numerical prediction in the context of neural networks and classification.

The k-nearest neighbor method (see → Section 5.1) can calculate numerical predictions. The other classification methods can be used for numerical prediction if the number of values of the numerical target attribute is not too large. Alternatively, the number of values of the metric target attribute can be reduced to a few by forming intervals, making classification methods applicable for prediction.

Linear regression is briefly discussed here, as it can be used not only as a data mining method but also in data preparation.

Linear regression

A regression function describes the *trend* or average relationship between *numerical attributes*. Linear regression assumes that there is a linear relationship between the data and the associated function values, that the relationship between the x and y values is *linear*, and that it can therefore be described by a straight line.

Given are sample data records X and the dependent target values $Y: E \subset X \times Y$. The relationship between these can now be described using the regression function $\hat{y} = f(x)$. The aim is to minimize the error between the actual target value and the calculated value. To prevent positive and

negative deviations from compensating each other, the squared error is examined:

$$\text{error} = \sum (y_i - \hat{y}_i)^2 \rightarrow \min \quad 3.1$$

Let's look at an example. Given the (x, y) values $(0, 3)$, $(1, 7)$, $(2, 6)$, $(3, 10)$. Linear regression is used to calculate a linear function, the straight line of which can be seen in → Figure 3.3. For other x values such as 4, 5, etc., the predicted value is then calculated from the equation of the straight line.

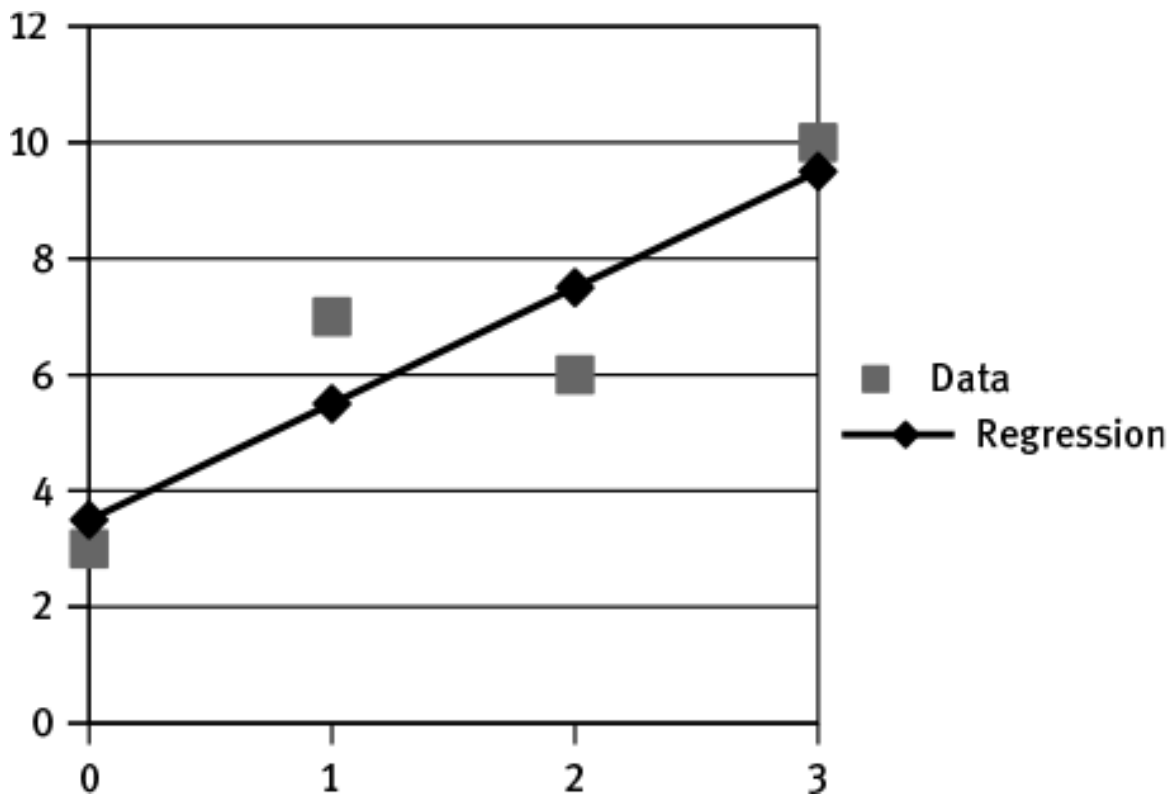


Fig. 3.3 Linear regression

The procedures for calculating the regression function are included in data mining tools and also in spreadsheet programs.

3.4 Association analysis

An association analysis (also known as *association rule mining*) identifies relationships between given attributes. A classic application is market basket analysis: What is bought together? Or more precisely: Which product is often bought together with another product? To this end, sales data is recorded in order to search for statements of the form:

Customers who buy product A often also buy product B.

In the future, the company will automatically offer product B when product A is purchased. Many online retailers already do this. But such analysis is also interesting in a supermarket, as it allows goods to be arranged accordingly.

Example 3.1

(Market basket analysis). Let's look at a fictitious market basket file.

Purchase	Milk	Bread	Water	Butter
1	x	x	x	x
2		x		x
3		x	x	x
4		x	x	

These four purchases show that butter was usually bought together with bread, namely in 75 % of cases. Conversely, bread is always bought together with butter.

We see that association rules do not necessarily *always* have to be 100 % correct; we will allow for exceptions. If item A is purchased, there is an X % probability that item B will also be purchased. This rule applies to Y % of purchases. Formally, we write this as follows:

$$A \rightarrow B \text{ [supp : } Y\% \text{] [conf : } X\% \text{]}$$

Association analysis is used to identify correlations between different items and analyze customer behavior. It is a predictive data mining method: the data is analyzed to identify patterns and predict the behavior of new data records. The relative frequency of an association rule is referred to as support *supp*, while confidence *conf* is introduced as a measure of trustworthiness of a rule (→ Section 9.2.1 and → 9.2.2).

Since the advent of barcode technology, it has been possible to document customer purchasing behavior without an unreasonable increase in workload. RFID technology will open up further possibilities in this area. The large amounts of data generated must be analyzed in order to find answers to questions such as: “How can I arrange my goods optimally?”, “How can I categorize my customers?” or “Which items should be removed from the product range?”.

The aim of a shopping basket analysis is, of course, to increase profits. One way to achieve this is by increasing customer satisfaction.

With its application in other areas, market basket analysis eventually developed into association analysis. This is an attempt to identify and describe regions or data areas in a database in which several values are highly likely to occur together.

Association analysis can therefore be seen as a generalization of market basket analysis. Possible applications include risk assessment in the insurance industry or analyzing the style of play of an opposing soccer team.

Association analysis can also be used for classification (→ Section 3.2), in which case only dependencies that contain information about the classification attribute are taken into account.

Which association analysis methods are covered in this book?

- **Apriori method** (→ Section 7.1)
- **Frequent pattern growth** (→ Section 7.2)

3.5 Text Mining

Text mining deals with the *analysis of text documents*. Unlike databases and web pages, texts are unstructured. Of course, texts also comply with

structural requirements such as grammar, but in the sense of our definition in → Section 2.1, they are unstructured.

A common task is the classification of documents, for example, by subject area or technical level. One approach is to first extract relevant terms from the document in order to classify it based on these terms. A second possible application is the quantification of the similarity of documents.

Text mining is performed in the same way as the data mining process models in → Section 1.2. Since only unstructured data is available, the first step is to extract the interesting information from the text documents. This could be keywords, the frequency distribution of terms, or an initial categorization or hierarchical grouping. Irrelevant words are filtered out and abbreviations are recognized.

It can also be useful to pre-structure a document into its constituent parts (title, chapter, section, sentence). In most cases, upper and lower case letters are treated as the same.

A problem is posed by the different morphological variants of a word. *Stemming* reduces words to their word stem, thus enabling a summary view of word variants. Semantic equality between different terms (synonyms) can and must also be detected. In German language there are two words for Saturday: “Sonnabend” has the same meaning as “Samstag”.

The result of this preprocessing is a reduced set of words (bag of words), which can also be weighted. The actual data analysis then takes place on the basis of these sets of words.

Often, a fixed set of keywords is specified and the number of occurrences of these keywords (after stemming) in the document is counted. A document can then be represented by a vector $(n_1, n_2, \dots, n_k)$ where n_i is the number of occurrences of the keyword i in the document.

This provides a well-structured representation that enables the use of distance measures and thus allows our methods to be applied.

Text mining is a field in its own right, which has given rise to many specific approaches and methods. For a more in-depth discussion of the topic of *text mining*, we refer to the literature, for example [→ FS07] and [→ Jo24].

3.6 Web Mining

Applications of data mining that use the Internet as a data source for pattern recognition are summarized under the topic of *web mining*. Depending on the content-oriented or usage-oriented analysis of the World Wide Web (WWW), the sub-areas *web content mining*, *web structure mining*, and *web usage mining* can be distinguished from one another.

Web content mining analyses textual or multimedia information in any format that builds the content of web pages on the WWW. For example, it is possible to analyze which topic is covered on a website or which political views a user holds (determined from forum posts).

Web structure mining examines links and link structures. This allows popular pages to be found. Search engines use the results of web structure mining. By examining whether people link to the same pages, the similarity between people can be determined.

Web usage mining focuses on the behavior of Internet users. In this form of web mining, data mining methods are applied to the log files of the web server in order to gain insights into the behavior patterns and interests of online users. A form of web usage mining in which the analysis is limited exclusively to the log files of the web server is referred to as *web log mining*. If other data sets are included in the pattern recognition process in addition to the log files, this form is referred to as *integrated web usage mining*.

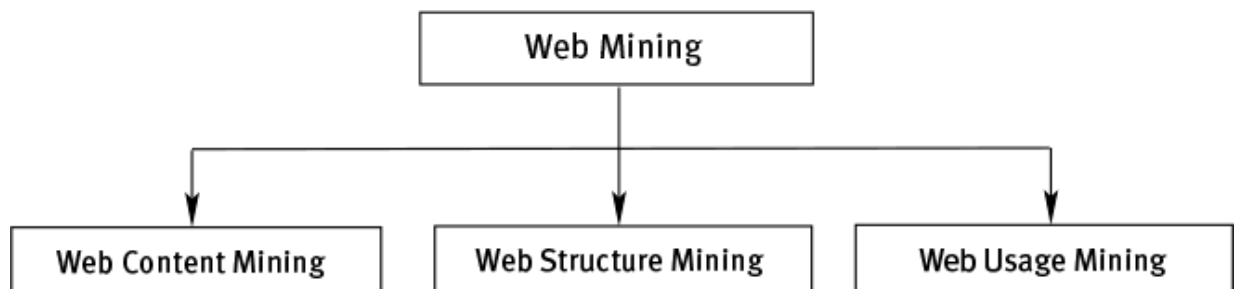


Fig. 3.4 Web mining

Web log mining in particular has gained importance due to the high significance of the Internet for companies. With an appealing website, a company can gain decisive competitive advantages over its competitors in the market. Companies collect usage data in log files about visits by (potential) customers to their websites. The data collected in this way is often not utilized sufficiently. From a business perspective, usage data relates to the behavior of market partners and is therefore important for supporting economic decisions. Management needs to know who visits the website and, more importantly, who visits the site and makes a purchase, or at what point potential customers leave the website.

Today, websites are seen as an investment and, like any other marketing investment, must justify their necessity. The more you know about how many customers visit your website, who they are, and what areas they are interested in, the more your website will benefit. The results of web mining, especially web usage mining, can help improve the website and, in combination with other marketing activities, increase sales.

Web log mining comprises data mining methods for evaluating Internet-based usage data (log files) and their use in supporting business decisions for the continuous improvement of websites and Internet offerings.

For an in-depth discussion of web mining, we refer to the literature, for example [→ LB01], [→ Liu11], and [→ Hon12].

4 Knowledge representation

Quality is inversely proportional to the time left for completion of the project.

Wright's first law of quality

After introducing the areas of application for data mining in the previous chapter, we will now discuss the forms in which the extracted knowledge can be *represented*. Subsequently, we will cover the methods of data mining.

Knowledge can be represented in different ways, for example explicitly as a rule or implicitly as a neural network. Rules themselves can be presented in various ways, for example as a decision tree, a table, or a logical formula. In → Section 3.3 on numerical prediction, we learned about another variant of knowledge representation: the result of a linear regression is a linear equation. This equation represents knowledge, in this case the functional relationship between the data.

The following sections introduce knowledge representation using decision tables, decision trees, and rules. In addition, possibilities for representing clusters and the role of individual objects as knowledge representatives are discussed.

4.1 Decision table

Decision tables are a form of representation for rules that can be used in different situations that require a decision: A programmer can use a decision table to specify his procedure, and an employee can use it to determine how to handle a complaint or an order. A decision table has four areas, see → Figure 4.1.

Tab. 4.1 Decision table for the weather example

Weather	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14
sunny	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N	N
overcast	N	N	N	N	N	Y	Y	Y	Y	N	N	N	N	N
rainy	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y
hot	Y	Y	N	N	N	Y	Y	N	N	N	N	N	N	N
mild	N	N	Y	Y	N	N	N	Y	N	Y	Y	Y	N	N
cool	N	N	N	N	Y	N	N	N	Y	N	N	N	Y	Y
humid	Y	Y	Y	N	N	Y	N	Y	N	Y	N	Y	N	N
windy	N	Y	N	Y	N	N	N	Y	Y	N	N	Y	N	Y
Play				X	X	X	X	X	X	X	X		X	
No play	X	X	X	X								X		X

In the original form, conditions are listed that can only be fulfilled or not fulfilled. The upper right part therefore consists only of the values Yes and No.

In the extended form, the attributes are specified and the values of these attributes are shown in the upper right area. Decision tables do not normally cover all possible cases. They show the cases to which actions can be assigned; cases for which knowledge is available. Decision tables are a means of describing complex situations using several easier-to-handle situations (columns).

Tab. 4.2 Extended decision table for the weather example

Weather	R1	R2	R3	R4	R5	R6	R7	R8	R9
outlook	sunny	sunny	sunny	sunny	overcast	rainy	rainy	rainy	rainy
temperature	hot	mild	mild	cool	-	mild	mild	cool	cool
humidity	high	high	normal	normal	-	-	high	normal	normal
windy	-	false	true	false	-	false	true	false	true
play	no	no	yes	yes	yes	yes	no	yes	no

A decision table in data mining is a tabular representation of the given data records, their conditions (inputs), and the corresponding results (outputs). This decision table can then be used for classification, at least for the cases that are recorded in the decision table.

Example 4.1

(Weather data). Decision → table 4.3 for the weather example is an alternative representation for decision → tables 4.1 and → 4.2. When will the

game be played? As can be seen from the example, a decision table does not necessarily cover every case. For example, there is no prediction for the situation (sunny, cool, high, false).

Tab. 4.3 Data mining decision table for the weather example

Outlook	Temperature	Humidity	Windy	Play
sunny	hot	high	false	no
sunny	hot	high	true	no
sunny	mild	high	false	no
sunny	mild	normal	true	yes
sunny	cool	normal	false	yes
overcast	hot	high	false	yes
overcast	hot	normal	false	yes
overcast	mild	high	true	yes
overcast	cool	normal	true	yes
rainy	mild	high	false	yes
rainy	mild	normal	false	yes
rainy	mild	high	true	no
rainy	cool	normal	false	yes
rainy	cool	normal	true	no

4.2 Decision trees

A *decision tree* is a form of representation in which the results of a condition are displayed in branches. These branches can in turn generate other branches. Decision trees are very good at visualizing the path to a prediction or decision. The graphical representation makes it easier to understand decisions, and the path also provides the reasoning behind the decision.

Decision trees can easily be converted into rules (→ Section 4.3). If you follow the path from the top node, called the root node or simply the root, to a result, called a leaf node, you get a rule similar to those in a decision table. → Figure 4.2, which shows a decision tree for the weather example (see → Appendix A.3), reveals the following rule:

IF outlook = sunny AND humidity = high THEN play = no

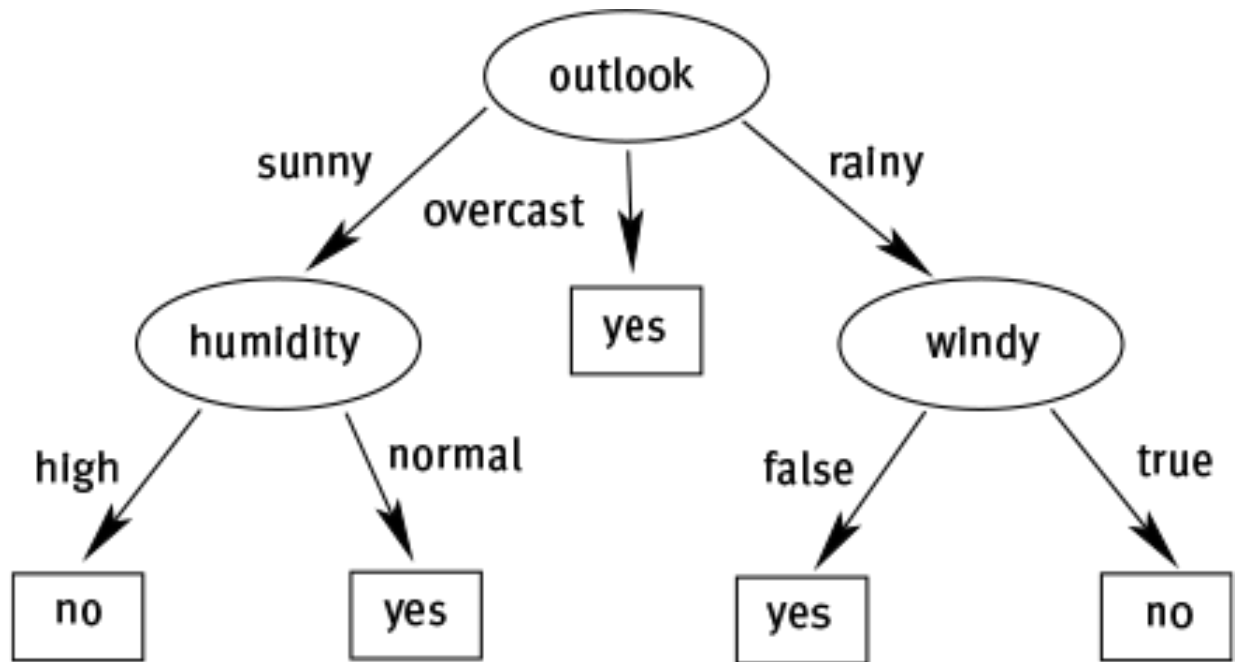


Fig. 4.2 Decision tree for the weather example

WEKA uses this form of representation to display the results of the J48 algorithm. WEKA uses this algorithm to generate the decision tree shown in → Figure 4.3 for the weather example.

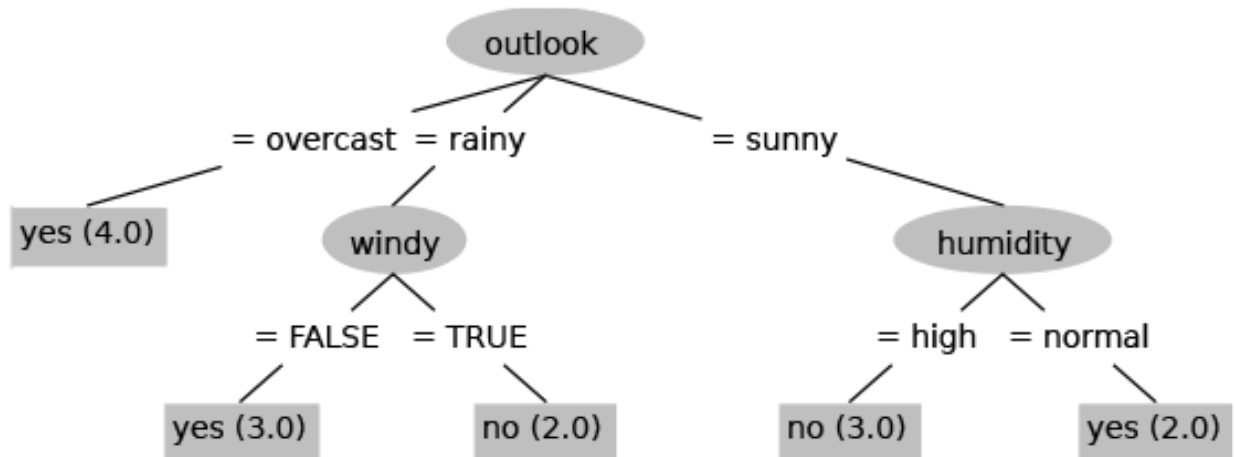


Fig. 4.3 Weka decision tree for the weather example

4.3 Rules

The representation of knowledge using logic, and in particular using rules, is an *explicit* representation of knowledge that is easy to understand and can also be used for documentation and visualization purposes. Rules, as explained in → Section 2.5, are logical formulas of the type:

IF condition THEN conclusion

Rules can be seen as the desired result of a data mining process. We can distinguish between classification rules and association rules. A *classification rule* describes the conditions under which a particular class is assigned:

IF outlook = rainy AND windy THEN play = no

A *classification rule* defines which class a data record is assigned to. The application reference required in the definition of *knowledge* becomes particularly clear in a rule: if the conditions apply, then an action is performed; in this case, assignment to a specific class.

Example 4.2

(Weather data). The following rules can be derived from the data in the weather example, which then represent the extracted knowledge:

IF outlook = sunny AND humidity = high THEN play = no

IF outlook = rainy AND windy = true THEN play = no

IF outlook = overcast THEN play = yes

IF humidity = normal AND windy = false THEN play = yes

IF none of the above THEN play = yes

In → Example 4.2, the first rule that “fits” is applied. So if rule 4 is tried, then the conditions of rules 1-3 were not met.

Classification rules always contain the target attribute in the conclusion part of a rule. In contrast, *association rules* describe relationships between arbitrary attributes. In addition, these rules are often accompanied by additional information that indicates the degree to which this association rule can be trusted. Due to the special nature of association rules, we devote a separate Section to this type of rule.

4.4 Association rules

An association rule is a rule that, as already mentioned, establishes relationships between arbitrary attributes. A typical example is the analysis of customer purchasing behavior.

Example 4.3

(Market basket analysis). In a supermarket (see → Example 3.1, p. 66), the shopping baskets of all customers are recorded at the checkout. Association rules can now be used to identify correlations between individual items. One result could contain the following findings:

- When washing powder is purchased, fabric softener is also purchased: *IF washing powder THEN fabric softener*
- When fish is purchased, meat is not purchased: *IF fish THEN NOT meat*
- When sparkling wine is purchased, chocolates are also purchased: *IF sparkling wine THEN chocolates*

The starting point is a set of *transactions* T , where a transaction $t \in T$ consists of a set of *items* I . The items can be regarded as the articles sold in a supermarket. In this case, a transaction is a purchase or a shopping basket.

The structure of association rules can be described as follows. An association rule consists of a condition M and a conclusion N . M and N are each conjunctions (AND operations) of items, which in turn represent the goods in the supermarket. The rule then takes the form

$$M \rightarrow N$$

with, for example, $M = \{I_1, I_2, I_3\}$ and $N = \{I_7\}$. The intersection of M and N must be empty.

The representation

$$\{\text{beer}, \text{chips}\} \rightarrow \{\text{TV guide}\}$$

can therefore be understood as an abbreviated notation for the rule

$$\text{IF beer} = \text{yes AND chips} = \text{yes THEN TV guide} = \text{yes.}$$

Example 4.4

(Weather data). Association rules for the weather example do not always contain the target attribute *play*. However, classification rules can be understood as association rules that contain the target attribute in the conclusion part.

IF temperature = cool THEN humidity = normal

IF humidity = normal AND windy = false THEN play = yes

IF outlook = sunny AND play = no THEN humidity = high

IF windy = false AND play = no THEN outlook = sunny

AND humidity = high

In most cases, these implications contain information about the frequency of their occurrence. Association rules are very rarely 100 % correct: not *all* customers comply with the rule

$$\{beer, chips\} \rightarrow \{TVguide\}$$

and buy accordingly, but only a certain percentage. In → Example 3.1, only 75 % of customers who buy bread also buy butter.

Consequently, we need interestingness measures that determine the quality of an association rule. We will discuss interestingness measures for association rules in detail in → Section 9.2. However, since these are necessary for understanding association rules, we will take a closer look at the interestingness measures *support* and *confidence*, which were already outlined in → Section 3.4.

The *support* of an item or an item set is the number of transactions that contain the item or item set as a subset, relative to the total number of transactions in set T . The support of an association rule is equal to the frequency of the union of the condition and conclusion ($M \cup N$) of the rule:

$$supp(M \rightarrow N) = P(M \cup N) \quad 4.1$$

The number of data records containing all attributes from $M \cup N$ is counted and divided by the total number of data records.

One observation is that $P(M \cup N)$ cannot be greater than both $P(M)$ and $P(N)$. The set union $\cup$ does not combine the set that satisfies M with the set that satisfies N . Rather, the *restrictions* that the set must satisfy are combined. $P(\{beer\})$ is the relative frequency of purchasing *beer*. $P(\{beer\} \cup \{chips\})$, on

the other hand, is the relative frequency of purchasing *beer* and *chips* together. $M \cup N$ obviously contains at least as many restrictions as M and N , respectively. Consequently, $P(M \cup N)$ can be at most as large as $P(M)$ or $P(N)$.

The second measure of interestingness is *confidence*. The confidence of an association rule is calculated from the ratio between transactions that contain both the premise and the consequence and transactions that contain only the premise. *Confidence* measures how often the rule actually applies in relation to the number of times it should have applied:

$$\text{conf}(M \rightarrow N) = \frac{\text{supp}(M \rightarrow N)}{\text{supp}(M)} \quad 4.2$$

Example 4.5

(Confidence and support). What are the support and confidence of the rule

IF temperature = cool THEN humidity = normal

IF temperature=cool THEN humidity=normal in $\rightarrow$ Example 4.1 (p. 72)?

$$\text{supp}(\text{temperature} = \text{cool} \rightarrow \text{humidity} = \text{normal}) = P(M \cup N) = \frac{4}{14}$$

$$\text{supp}(\text{temperature} = \text{cool}) = P(M) = \frac{4}{14}$$

$$\text{conf}(M \rightarrow N) = \frac{\text{supp}(M \rightarrow N)}{\text{supp}(M)} = \frac{\frac{4}{14}}{\frac{4}{14}} = 1$$

The confidence of the rule is 1, the rule is absolutely certain. The support is $\frac{2}{7}$

Threshold values

In order to separate valuable association rules from less valuable ones, threshold values for confidence and support are introduced, which must not be fallen below for an association rule. These are referred to as $\text{conf}_{\min}$ and $\text{supp}_{\min}$ and are usually defined by the user.

In most cases, the greater the support and confidence, the more valuable the association rule. There may be exceptions. For example, rules such as

$$\{\text{Personlives}\} \rightarrow \{\text{Personbreathes}\}$$

trivially have a high confidence level and are nevertheless uninteresting.

Types of association rules

In addition to the original association rules, also known as *Boolean association rules*, a number of variations have been developed to eliminate certain disadvantages. We present the following types:

- *Hierarchical* association rules (taxonomies)
- *Temporal* association rules (sequence analysis)
- *Quantitative* association rules
- *Fuzzy* association rules

Why do we consider other forms of association rules? On the one hand, we want to increase the significance of rules. On the other hand, numerical attributes often present the problem that a value lies just outside an interval and is therefore not considered to belong to that interval, which is mathematically correct but often makes no sense in practice.

Hierarchical association rules

In *hierarchical association rules*, several terms (items) are grouped together under a generic term (for example, individual products into product groups or categories). It is possible for a term to have several generic terms. For example, a term can belong to the generic term *special offer* as well.

The advantage of hierarchical association rules is that combining terms increases the support of the rule and more interesting rules are found. Rules are now found at a more abstract level.

Example 4.6

(Grouping of items). To establish hierarchical association rules, individual items are grouped together. Instead of creating many rules with the items on the left side of the table, only a few are now sufficient.

Items	Generic term
{knife, fork, spoon}	cutlery
{bread, milk, cheese}	food
{one-two, cross}	attack
...	...

Quantitative association rules

Simple and hierarchical association rules do not provide any information about the quantity of a term. They only indicate whether the term appears in the transaction. *Quantitative association rules* also take into account the quantity of the term and enable a more precise distinction to be made. This is useful for numerical terms such as number, age, or price. The procedure is as follows:

1. Dividing the value range into *intervals*,
2. Creating a *new term* for each interval,
3. Replacing the original terms by the new ones.

Example 4.7

(Interval formation). Given the following database:

Age	Number of children	Income
23	0	2000
28	1	2500
34	0	4500
45	2	3500
65	5	4000

The term *age* is divided into three intervals: *under 30*, *30 to 50*, and *over 50*. The term *number of children* is divided into two intervals: *no children or one child*, and *more than one child*. Income is also divided into two intervals: *up to 3000* and *more than 3000* (see binary coding in → Section 8.2.5). A new database is created based on this data.

Age1	Age2	Age3	Children1	Children2	Income1	Income2
1	0	0	1	0	1	0
1	0	0	1	0	1	0
0	1	0	1	0	0	1
0	1	0	0	1	0	1
0	0	1	0	1	0	1

A possible quantitative rule would then look as follows:

IF age $\in [0, 29]$ *AND income* $\in [0, 2999]$ *THEN children* = 0/1

Fuzzy association rules

Fuzzy association rules are an advancement of quantitative association rules. The interval limits are now fluid rather than rigid. For example, the term *age* is divided into *young*, *middle-aged*, and *old*. People up to the age of 25 are classified as *young*, those aged 35 and above as *middle-aged*, and those in between belong to both groups. Due to the fluid interval limits, the support of a rule is greater than with quantitative association rules, as terms that were just outside the interval limits in quantitative association rules are now also taken into account.

Fuzzy rules are particularly suitable for data analyses that involve outliers or measurement errors, such as in physics.

Example 4.8

(Quantitative association rules). A call center plans to store data on incoming calls. This data includes, among other things, the time at which the call was answered. Suppose the center's management wants to sort the calls by time of day. The quantitative approach divides the 24 hours of the day into intervals, such as *night*, *morning*, *afternoon*, and *evening*. The night interval ends at 6 a.m., followed by the morning interval. The morning ends at 12 p.m. and transitions into the afternoon, and so on.

The characteristic feature of this approach is that overlaps can occur. For example, there may be a group of callers who call between 6 a.m. and 12 p.m. to order bread rolls. This results in a rule of the form:

$$IF \text{ time} = \text{morning} THEN \text{order} = \text{bread rolls}$$

The disadvantage of this approach is that there may be customers who call shortly before 6 a.m. or shortly after 12 p.m. to order bread rolls. Since the time of these calls no longer falls within the specified interval, these calls no longer belong to the *morning* customer group, regardless of whether they meet all other characteristics of this group. The support for the above rule decreases.

Fuzzy rules counteract this behavior. Instead of fixed intervals, degrees of membership are used. A call at 11 a.m. can be assigned to both the *morning* group and the *afternoon* group. The assignment is made using methods from fuzzy logic (see [→ LC23]). The main difference between fuzzy and quantitative association rules lies in the way the rules are generated; in terms of further behavior, both forms are similar.

Temporal association rules

In the case of *temporal association rules* or sequence analysis, in addition to the information that a term occurred in a transaction, the time or sequence is also recorded. This means that, unlike normal association analysis, this analysis is a *time sequence analysis*.

The aim of sequence analysis is to find sequences that show a minimum level of support. For example, if many crates of beer are bought every Friday and the next day sales figures for headache tablets skyrocket, there is a temporal relationship that can be taken into account. In temporal databases, data is stored in a temporal context. The association rules can be understood as *snapshots* of the changing relationships between the data. This makes it possible to observe and explore changes and fluctuations in these relationships.

A typical application for sequence analysis is log file analysis, which examines the behavior of visitors to a website.

4.5 Instance-based representation

In *instance-based representation*, all data records in the training set are simply stored. We can regard this training set as knowledge because it can be interpreted as a decision table (see → Section 4.1). Each data record embodies a rule:

If the attribute values correspond to the values of an existing example,
then the class to be assigned corresponds to the class of this data record.

Using all given data records as a representation of knowledge is, in a sense, equivalent to memorization: no generally applicable rules are derived, but rather all objects are memorized individually, i.e., stored.

4.6 Representation of clusters

If a set is divided into subsets whose objects are more similar to each other than to the objects in the other subsets, these subsets are referred to as *clusters*. A cluster can be represented

- as a *set* of objects assigned to it,
- as a *vector of the mean values* of the attribute values of its objects (centroid),
- by a *typical representative* of the cluster elements (medoid), or
- by *probabilities*.

When probabilities are used, they indicate the probability with which an individual can be assigned to a cluster.

→ Figure 4.4 shows both the centroid (the mean value or center of gravity) and a possible medoid for an example cluster.

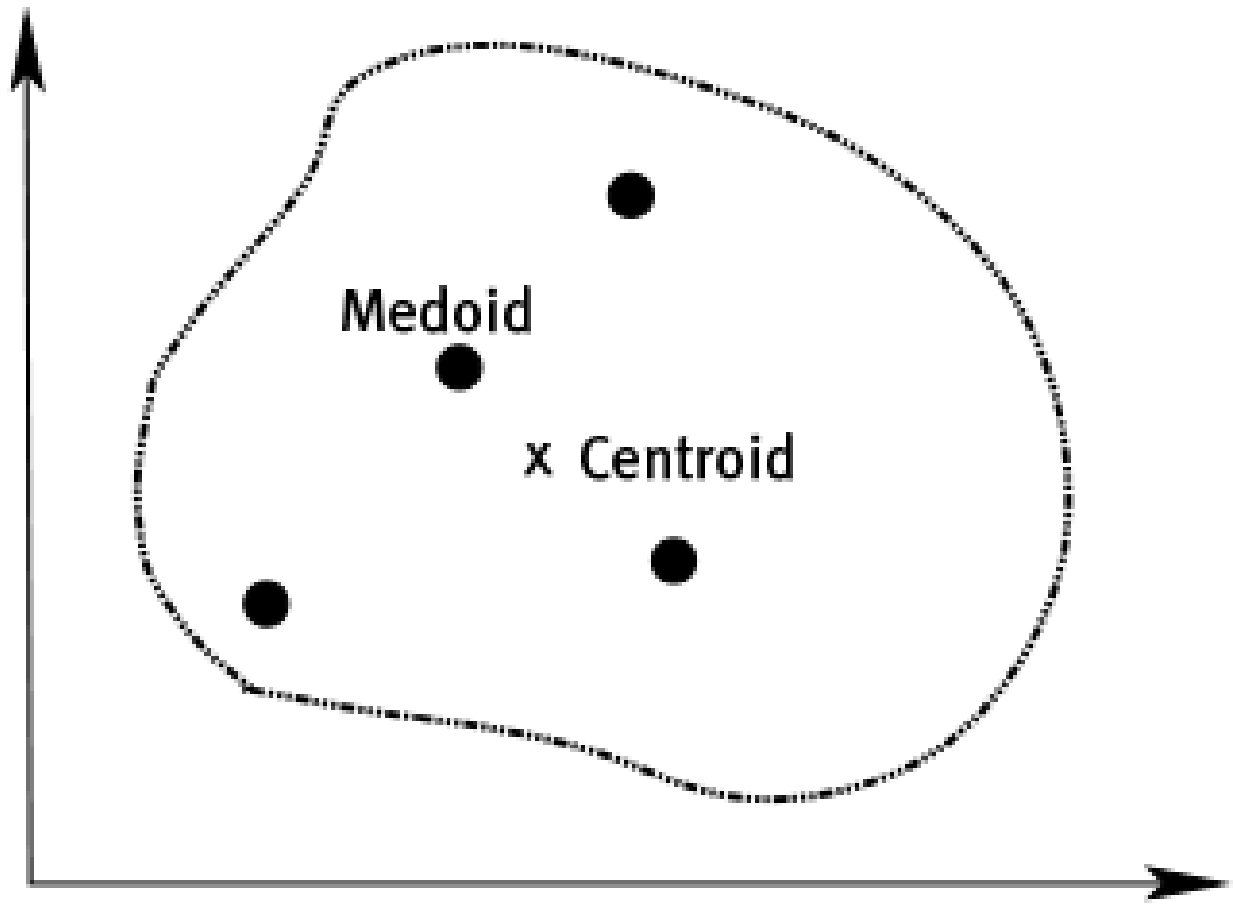


Fig. 4.4 Centroid and medoid

The *centroid* can be an artificially created instance; it does not have to occur in the data set. In contrast, a *medoid* is always an element of the data set.

The *median* is also a suitable cluster representative (see → Section 6.3).

Using the k-means algorithm (→ Section 6.2), WEKA achieves the following clusters for the weather example:

```
kMeans
Number of iterations: 4
Within cluster sum of squared errors: 26.0
```

Cluster centroids:					
Cluster 0	Mean/Mode:	sunny	mild	high	FALSE yes
	Std Devs:	N/A	N/A	N/A	N/A
Cluster 1	Mean/Mode:	overcast	cool	normal	TRUE yes
	Std Devs:	N/A	N/A	N/A	N/A
Clustered Instances					
0	10 (71%)	1	4 (29%)		

4.7 Neural networks as knowledge stores

Looking at the structure of a neuron and a neural network (see → Section 2.4), the question arises:

Where is knowledge stored in an artificial neural network?

To answer this question, we examine (for example in → Figure 4.5) which entities are variable in an artificial neural network.

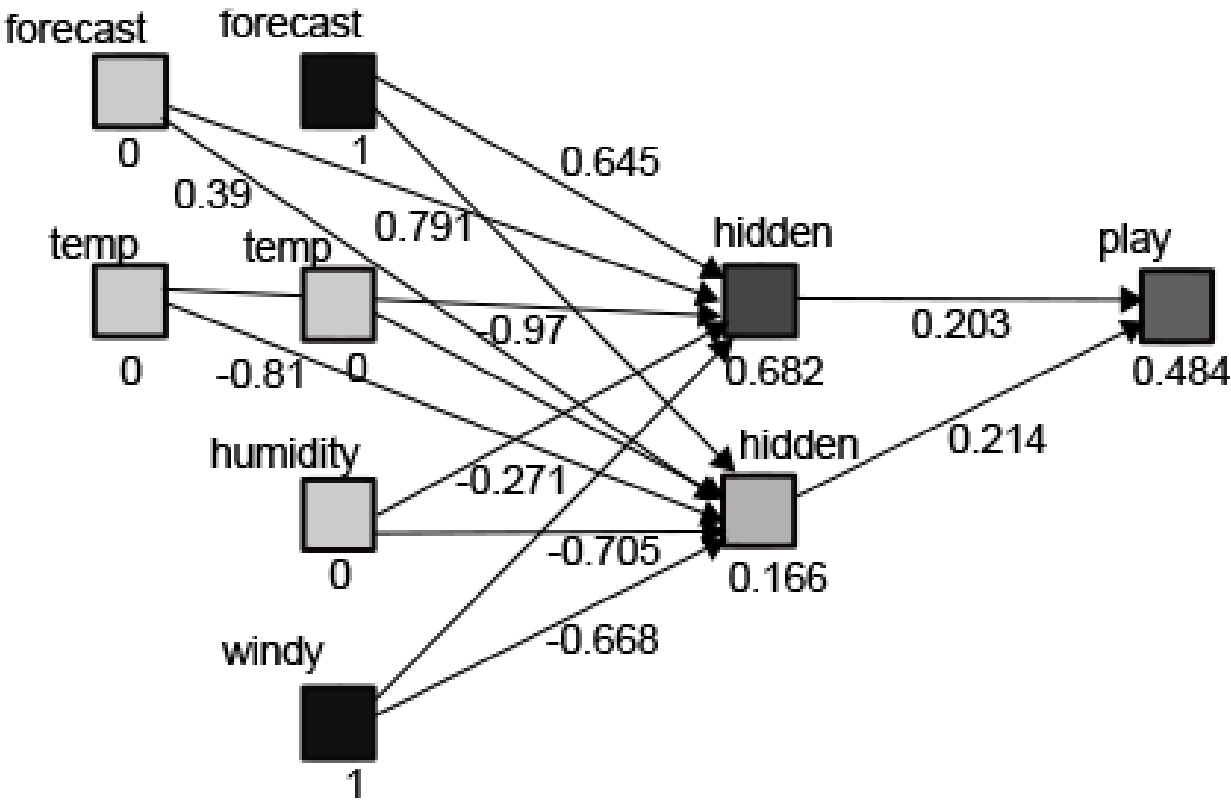


Fig. 4.5 Artificial neural network as a knowledge repository

During the training process, *connection weights* and *threshold values* are changed. The amount of connection weights and threshold values determines the behavior of the network. This is the knowledge that is acquired during the training process, i.e., by processing the training data, and thus defines the behavior of the network.

An artificial neural network also represents the knowledge of the developer who defines the network architecture, namely:

- the number of neurons in the input layer,
- the number of neurons in the output layer,
- the number and size of hidden layers, and
- the number and type of connections between the neurons.

A neural network represents knowledge *implicitly*, analogous to knowledge storage in the natural neural network. We cannot explicitly describe how we recognize objects, maintain our balance, walk, swim, or ride a bike. These are all examples of human abilities that we have acquired and internalized over the course of our lives. We do this “automatically”, without thinking, and rely on our acquired skills every day.

Similarly, when using an artificial neural network, we must trust that the acquired behavior will deliver the desired results. We cannot expect an artificial neural network to justify its decisions. → Figure 4.5 shows the trained network for the weather example.

After having examined the application classes and the possibilities for representation on a computer, we now turn to the actual core, the data mining methods.

5 Classification

A carelessly planned project takes three times longer to complete than expected; a carefully planned project takes only twice as long.

Golub's Second Law of Computerdom

Classification is one of the most important applications in data mining and an example of *supervised* learning, see → Section 2.6: Based on given examples whose class assignment is known, a classification of new data records whose class assignment is unknown is performed, see → Section 3.2.

There are two approaches here: In *instance-based* methods, new data is classified directly using the given sample data set. These methods do not develop a model and are referred to as *lazy learners*. Instance-based methods are quite simple. All objects in the training set are stored. To classify an unknown object, the most similar object from the known set is then searched for and its class is predicted.

A second class of methods calculates a *model* based on the known examples. New data records are classified using only the developed model; the sample data is no longer needed. These methods are *eager learners*.

Some of the workflows for developing the models presented in this chapter, as well as the associated data, are available for download at the book's website, [→ https://www.fww.hswismar.de/datamining/](https://www.fww.hswismar.de/datamining/).

5.1 The k-nearest neighbor method

The k-nearest neighbor method (kNN) is an instance-based method. It requires a distance measure for the data to be classified.

Most implementations require real-valued attributes. However, nominal or ordinal attributes can also be used with suitable distance measures.

The learning step of the method is simple: all sample objects are stored. Unknown objects are classified by calculating the *similarity* (see → Section 2.3) of their descriptive attribute values to those of the objects already stored. The *k* objects that are most similar to the new object are used to predict the class of the new object. The class that occurs most frequently among the *k* nearest neighbors is predicted for the object to be classified.

→ Figure 5.1 illustrates the procedure. Suppose we have two-dimensional data records as points belonging to two different classes.

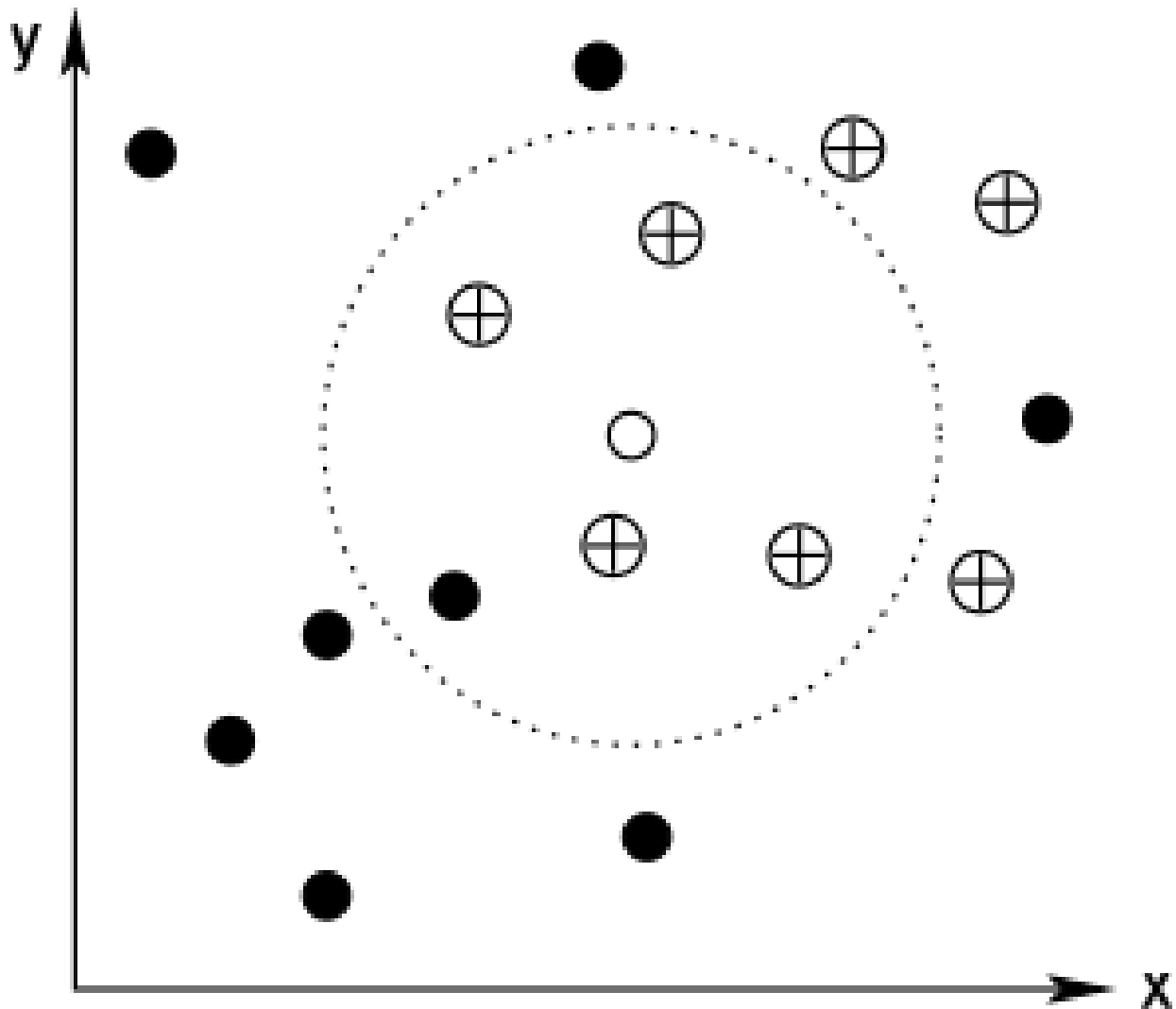


Fig. 5.1 K-nearest neighbor, example 1

Objects marked with $\oplus$ belong to one class, while objects belonging to the other class are marked with $\bullet$. The class affiliation of the object $\circ$ is unknown. Based on the class affiliation of the other objects, the new object should be assigned to one of the two classes.

If we take the 5 nearest neighbors and choose the Euclidean distance as our distance measure, the result is class $\oplus$, since class $\oplus$ wins against $\bullet$ by a ratio of 4:1. However, problems can arise, as illustrated by changing the arrangement of the points ($\rightarrow$ Figure 5.2). With $k = 1$, illustrated by the solid circle, $\text{class}(\circ) = \bullet$, whereas with $k = 5$ (dashed circle), $\text{class}(\circ) = \oplus$. The choice of k can therefore be decisive for the result. In practice, therefore, several variants with different k are usually calculated and the respective predicted classes are compared.

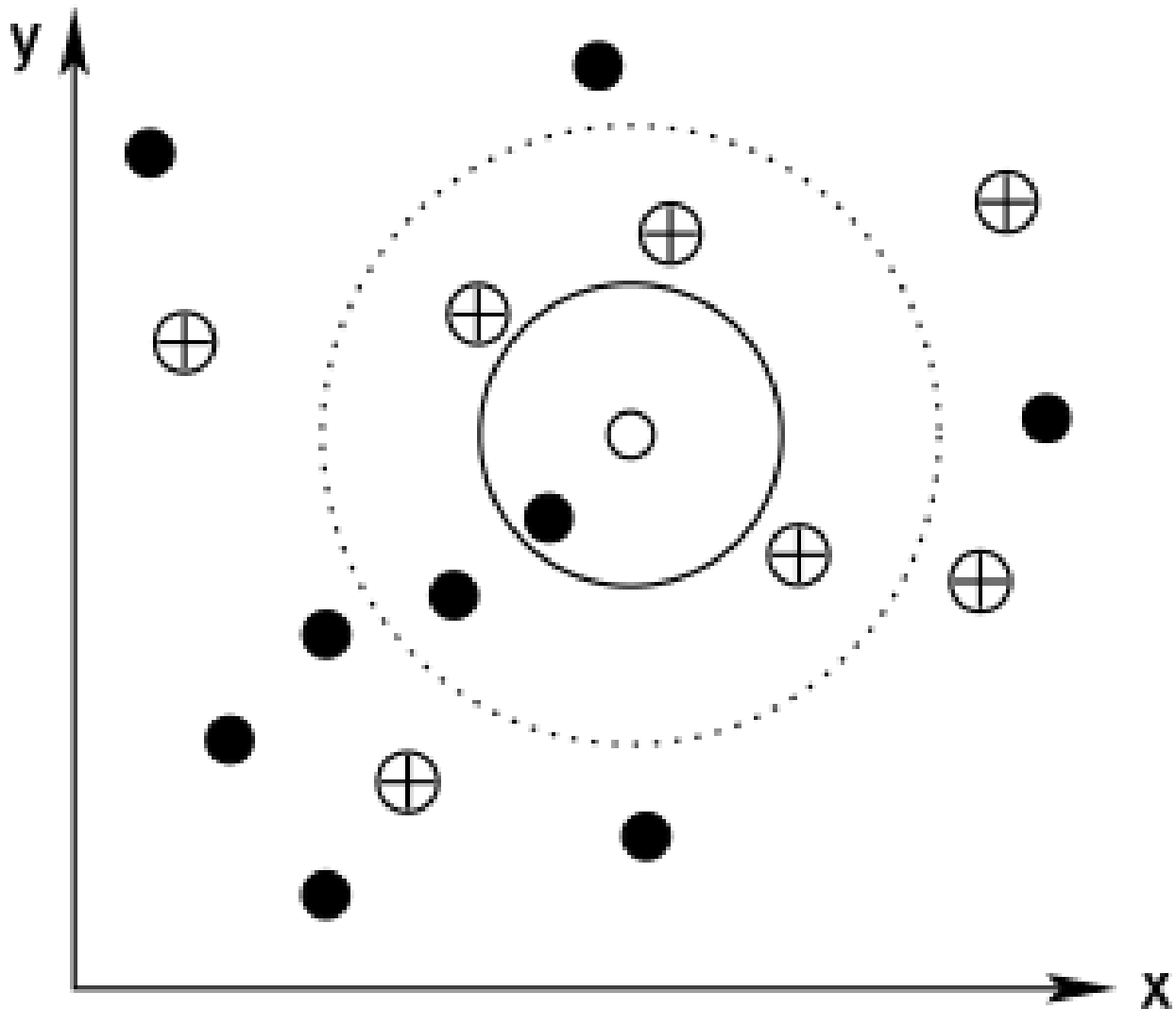


Fig. 5.2 K-nearest neighbor, example 2

The complexity of the method is determined not by a learning step, but by the classification step: the distances between all sample data records and the object to be classified must be calculated.

The kNN method does not generate a model, but classifies directly using the given sample data set: kNN is a *lazy learner*. The kNN method is related to case-based reasoning and locally weighted regression.

5.1.1 The k-nearest neighbor algorithm

In the strict sense of the word, there is no learning in kNN. But the preparation for further classifications can be classified as “learning” as follows (let f be the classifier sought):

Store each training example $(x, f(x))$ in a list.

Listing 5.1 (k-nearest neighbor, learning step).

For a new data record x' , the function value $f(x')$ must now be determined. Depending on the data types for x and $f(x)$, different types of functions f can be distinguished, whose classification steps differ slightly from one another.

5.1.1.1 Discrete function

Let $V = \{v_1, v_2, \dots, v_m\}$ be the finite set of values that the target attribute can take, and let $f: \mathbb{R}^n \rightarrow V$ be the discrete¹ function to be learned. Let y be the example to be classified, then the k -nearest neighbor algorithm has the following form:

1. For all x_i in the set of training examples, calculate the similarity to y .
2. Select those k examples $x_1, x_2, \dots, x_k$ that are most similar to y .
3. The class for y is calculated as follows:

$$\text{class}(y) = \max_{v \in V} \sum_{p=1}^k \delta(v, f(x_p)) \quad \text{with} \quad \delta(a, b) = \begin{cases} 1, & \text{if } a = b \\ 0, & \text{else} \end{cases} \quad 5.1$$

Listing 5.2 (k -nearest neighbor, classification step, discrete)

The *class* function calculates the class that occurs most frequently among the k training examples. This means that y is assigned to the class that is *most frequently* represented among the k most similar examples.

In the definition of the function *class*, we can use $f(x_p)$ even though we do not know the function. However, we do know the value $f(x_p)$, because it is given by the sample set $\{(X_i, f(X_i)) \mid i = 1, \dots, k\}$.

Example 5.1

(k -nearest neighbor, income, real-valued attributes). The data given in → Table 5.1 on the living conditions of eight people is examined. Let the attribute to be predicted be *income*: *high*, *medium*, *low*. The income to be predicted is for a person with the following characteristics:

- 26years old,
- married (1),
- academic (1),
- without home ownership (0).

Tab. 5.1 Income table

No.	Age	Married	Home ownership	Academic	Income
new	26	1	0	1	?
1	59	1	1	1	high
2	55	1	0	0	low
3	40	0	0	0	low
4	37	1	1	1	high
5	26	0	0	0	low
6	24	1	0	0	medium
7	22	1	1	1	medium
8	53	0	1	0	high

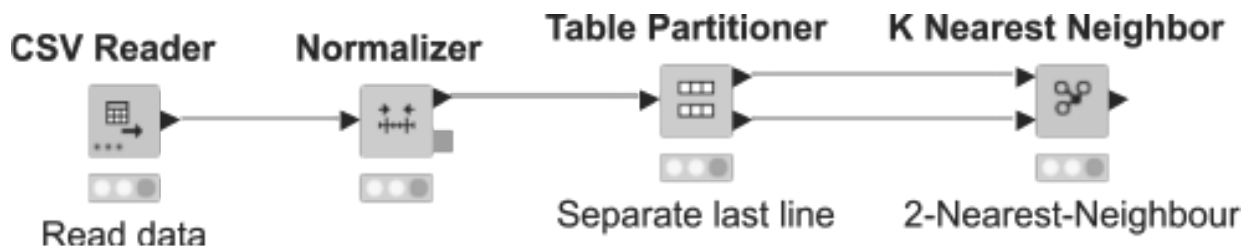
First, the k data records most similar to the new data record are determined. We select $k = 2$ and determine the two most similar data records without taking the *income* attribute into account. The Euclidean distance is used as the distance measure.

We should not apply the k-nearest neighbor method here without preprocessing the data, as otherwise the Euclidean distance of the age data will dominate all other attributes. We therefore first normalize the values of the *age* attribute, see → Section 8.2.5. Then the distances of data records 1—8 from the new data record are recalculated, see → Table 5.2.

Tab. 5.2 Income table with normalized data

No.	Age	Married	Home ownership	Academic	Income	Distance to new
new	0.11	1	0	1	?	-
1	1.0	1	1	1	high	1.34
2	0.89	1	0	0	low	1.27
3	0.49	0	0	0	low	1.464
4	0.41	1	1	1	high	1.043
5	0.11	0	0	0	low	1.414
6	0.05	1	0	0	medium	1.001
7	0.0	1	1	1	medium	1.006
8	0.84	0	1	0	high	1.879

Data records 6 and 7 are the closest neighbors, so we predict a medium income. KNIME correctly calculates the class *medium*, see → Figures 5.3 and → 5.4.

**Fig. 5.3** Workflow of k-nearest neighbor for the income example

File Edit Hilite Navigation View						
Table "default" - Rows: 1		Spec - Columns: 6		Properties		Flow Variables
Row ID	D Age	I Married	I House	I Acade...	S Income	S Class ...
9	0.108	1	0	1	?	medium

Fig. 5.4 Result of k-nearest neighbor for the income example

Normalization to a different interval, for example $[0, 2]$, is also possible. This changes the influence of the *age* attribute. The larger the interval, the greater the distances and the greater the influence of the attribute, see → Section 8.2.5.

In the next example, we show that the k-nearest neighbor method can also work with nominal data.

Example 5.2

(k-nearest neighbor, income, nominal attributes). → Table 5.3 again shows the data records on the living conditions of the eight people, but now with nominal attributes.

Tab. 5.3

Income table with nominal data

No.	Age	Married	Home ownership	Academic	Income
1	old	yes	yes	yes	high
2	old	yes	no	no	low
3	medium	no	no	no	low
4	medium	yes	yes	yes	high
5	young	no	no	no	low
6	young	yes	no	no	medium
7	young	yes	yes	yes	medium
8	old	no	yes	no	high

Let the attribute to be predicted be the *income* again: *high*, *medium*, *low*. We want to predict the income for a young, married academic who does not own a home. First, we determine the k data records most similar to the new data record. We take $k = 2$ and determine the two most similar data records, again without considering the *income* attribute. As a distance measure, we now use the Hamming distance (see → Section 2.3), which counts the number of *different* attribute values, see → Table 5.4.

Tab. 5.4

Income table with Hamming distance

No.	Age	Married	Home	Academic	Distance
new	young	yes	no	yes	
1	old	yes	yes	yes	2
2	old	yes	no	no	2
3	medium	no	no	no	3
4	medium	yes	yes	yes	2
5	young	no	no	no	2
6	young	yes	no	no	1
7	young	yes	yes	yes	1
8	old	no	yes	no	4

In the two most similar data records (6 and 7), the *income* that is most frequently represented is counted: this is again the *medium* income.

5.1.1.2 Real-valued function

What changes when we are not dealing with discrete classes, but with a real-valued target attribute? For real-valued functions $f: \mathfrak{R}^n \rightarrow \mathfrak{R}$, the k-nearest neighbor algorithm differs from the discrete case only in that the mean value of the target attribute values of the selected k examples is now returned.

1. For all x_i in the set of training examples, calculate the similarity to y .
2. Select those k examples $x_1, x_2, \dots, x_k$ that are most similar to y .
3. The target attribute value for y is calculated as follows:

$$f' \left(y \right) = \frac{\sum_{p=1}^k f(x_p)}{k}$$

5.2

Strictly speaking, with this variant of k-nearest neighbor, we are leaving the realm of classification and engaging in numerical prediction; see → Section 3.3.

Listing 5.3 (k-nearest neighbor, classification step, real-valued).

Example 5.3

(k-nearest neighbor, income, real target attribute). We change the *income* column in Example 5.2 and enter numbers there.

No.	Age	Married	Home ownership	Academic	Income
1	old	yes	yes	yes	4050
2	old	yes	no	no	950
3	medium	no	no	no	1005
4	medium	yes	yes	yes	3890
5	young	no	no	no	800
6	young	yes	no	no	2300
7	young	yes	yes	yes	2700
8	old	no	yes	no	3780

Let $k = 2$. We want to predict the income for a young, married academic who does not own a home. What changes when determining the two most similar data records? Nothing, because the distance has nothing to do with the target attribute. The two most similar data records therefore remain data records 6 and 7. Their incomes are averaged, and we get 2500.

5.1.2 A refined algorithm

The uncertainties shown in → Figure 5.2 (the possible fluctuations in the result for different k) can be counteracted by refining the k-nearest neighbor algorithm.

One weakness of the k-nearest neighbor algorithm discussed so far is that all k nearest neighbors contribute equally to the classification result. However, if the basic assumption that a *small distance* implies a *high similarity* is valid, then it seems reasonable to assign a weight to each individual data record according to its distance from the new data record. This approach also makes it possible to use all training examples to determine the result, because those with a large distance hardly contribute to the result. This approach is known as *Shepard's method*. However, this method implies a higher computational effort in the classification step.

5.1.2.1 Discrete function

Let $V = \{v_1, v_2, \dots, v_m\}$ be the finite set of values that the target attribute can take, and let $f: \mathbb{R}^n \rightarrow V$ be the function to be learned.

The inverse of the square of the distance between the training data record and the new data record is chosen as the weight.

The classification step refined by distance weighting is therefore as follows:

1. For all x_i in the set of training examples, calculate the similarity to y .
2. Select those k examples $x_1, x_2, \dots, x_k$ that are most similar to y .
3. The class for y is calculated as follows:

$$\text{class}\left(y\right) = \begin{cases} f(x_i) & \text{if } y = x_i \text{ for at least one } i \\ \max_{\nu \in V} \sum_{p=1}^k \omega_p \cdot \delta\left(\nu, f\left(x_p\right)\right) & \text{else} \end{cases} \quad 5.3$$

$$\text{with } \delta(a, b) = \begin{cases} 1, & \text{if } a = b \\ 0, & \text{else} \end{cases} \text{ and } \omega_p = \frac{1}{\text{dist}(y, x_p)^2}$$

Listing 5.4 (Shepard's method, classification step, discrete).

In this algorithm, we have already taken into account the special case that our new data record y is already contained in the given sample set as x_i . If $y = x_i$, then the distance between y and x_i is of course zero and the weight w_i is then undefined. In this case, we predict the class of x_i for the data record y .

5.1.2.2 Real-valued function

If we do not have discrete classes but a real-valued target attribute, we can again average the numerical values, now weighted, of course.

1. For all x_i in the set of training examples, calculate the similarity to y .
2. Select those k examples $x_1, x_2, \dots, x_k$ that are most similar to y .
3. The target attribute value for y is calculated as follows:

$$f'(y) = \begin{cases} f(x_1) & \text{if } y = x_i \text{ at least for one } i \\ \frac{\sum_{p=1}^k w_p \cdot f(x_p)}{\sum_{p=1}^k w_p} & \text{else, with } \omega_p = \frac{1}{\text{dist}(y, x_p)^2} \end{cases} \quad 5.4$$

Listing 5.5 (Shepard's method, classification step, real-valued).

Notes

- The kNN algorithm has the advantage of being very simple. However, if the amount of data is large, runtime problems can occur because the distance of the data record to be classified is calculated for *all* data.
- For large k ($k \gg 1$), the k-nearest neighbor algorithm also works well with noisy training data, i.e., real-valued data with a spread.
- In contrast to the decision tree, for example, *all* attributes are included in the calculation. This means that the algorithm becomes significantly less reliable if some of the attributes are irrelevant for determining similarity. This problem can be mitigated by assigning weights to the attributes. This can be achieved by using a distance measure with weighted attributes. The problem can also be circumvented by excluding attributes that are likely to be irrelevant from the outset.

- The selection of training data deserves some attention; it should cover the solution space as uniformly as possible.
- Choosing a suitable k is crucial for the procedure. As already discussed, you should start the k -nearest neighbor procedure with different k values and then predict the class that wins most often.
With two classes, you avoid an even k and thus a tie in the prediction.
- What do we do if, for example, the 5 nearest neighbors are not uniquely determined? It may happen that we have two neighbors with a distance of 1 and five neighbors with a distance of 2. Then it is not clear which are the 5 nearest neighbors. In this case, we have no choice but to increase k to 7.
Of course, if all 7 neighbors belong to the same class, the problem is irrelevant.

Task 5.1

Task 5.1 (k -nearest neighbor, restaurant data).

Consider the data set given in → Table 5.5 (see → Task 5.7). Classify the last 3 data records based on the given 12 data records using k -nearest neighbor. Test the method with different k . Which distance function would you suggest?

The table contains data records with the following attributes:

- Alternative: Is there another suitable restaurant nearby? {yes, no}
- Fri/Sat: Is it Friday or Saturday? {yes, no}
- Hungry: Am I hungry? {yes, no}
- Guests: How many people are in the restaurant? {none, a few, full}
- Reservation: Have I made a reservation? {yes, no}
- Type: What kind of restaurant is it? {French, Chinese, Italian, Burger}
- Waiting time: How long does the restaurant estimate the wait will be? {0-10, 10-30, 30-60, >60}
- Waiting (target attribute): Am I waiting? {yes, no}

Tab. 5.5 Restaurant example

Alternative	Fri/Sat	Hungry	Guests	Reservation	Type	Waiting time	Waiting
yes	no	yes	a few	yes	French	0-10	yes
yes	no	yes	full	no	Chinese	30-60	no
no	no	no	a few	no	Burger	0-10	yes
yes	yes	yes	full	no	Chinese	10-30	yes
yes	yes	no	full	yes	French	>60	no
no	no	yes	a few	yes	Italian	0-10	yes
no	no	no	none	no	Burger	0-10	no
no	no	yes	a few	yes	Chinese	0-10	yes
no	yes	no	full	no	Burger	>60	no
yes	yes	yes	full	yes	Italian	10-30	no
no	no	no	none	no	Chinese	0-10	no
yes	yes	yes	full	no	Burger	30-60	yes
yes	no	yes	a few	no	French	30-60	
yes	yes	yes	full	yes	Chinese	10-30	
no	no	no	none	no	Burger	0-10	

Task 5.2

Task 5.2 (*k*-nearest neighbor, weather example 1).

Apply the *k*-nearest neighbor method to the weather example (→ Appendix A.3, nominal variant) and predict whether or not there will be a match in the following situations. Which distance measure will you choose? Apply the method with different values of *k*.

outlook=sunny, temperature=mild, humidity=normal, windy=false
outlook=sunny, temperature=mild, humidity=normal, windy=true
outlook=rainy, temperature=hot, humidity=normal, windy=false

Task 5.3

Task 5.3 (*k*-nearest neighbor, weather example 2).

Apply *k*-nearest neighbor to the weather data (→ Appendix A.3, numerical variant) and predict whether or not there will be a match in the following situations. Which distance measure do you choose? Can you use the data in its original format, or do you have to encode the data differently?

outlook=sunny, temperature=78, humidity=83, windy=false
outlook=rainy, temperature=69, humidity=94, windy=true

5.2 Decision tree learning

5.2.1 Creating a decision tree

We have already seen a decision tree in → Figure 4.2 (→ Section 4.2, p. 73). In a decision tree, the first question asked is about the value of *one* attribute. In → Figure 4.2, this is the question about the value of *outlook*. Depending on the value of the attribute, either the class is predicted directly, as is the case for *outlook=overcast*, or further attributes are queried.

To generate a decision tree, an attribute $a \in A$ must first be selected as the root, based on the attribute set A and the example set E . Let ω_a be the set of all values that the attribute a can take. For each value $\omega \in \omega_a$ of this attribute, the set $E^\omega \subseteq E$ is formed, for which $\forall e \in E^\omega : \omega_a(e) = \omega$ applies. Furthermore, an edge marked with this value is placed at the root. If $E^\omega = 0$, the edge is closed with NIL. If *all* examples from E^ω are in the *same* class, the edge is closed with a leaf marked with the corresponding class. In any other case, another decision tree is generated, whose root is appended as a node to the end of the edge. This new subtree is now generated in a similar way. However, the reduced attribute set $A' = A \setminus \{a\}$ and the reduced example set E^ω are used to form the subtree.

For clarification, let us once again consider the weather data, see → Table 5.6.

Tab. 5.6 Weather example

Day	Outlook	Temperature	Humidity	Windy	Play
1	sunny	hot	high	false	no
2	sunny	hot	high	true	no
3	overcast	hot	high	false	yes
4	rainy	mild	high	false	yes
5	rainy	cool	normal	false	yes
6	rainy	cool	normal	true	no
7	overcast	cool	normal	true	yes
8	sunny	mild	high	false	no
9	sunny	cool	normal	false	yes
10	rainy	mild	normal	false	yes
11	sunny	mild	normal	true	yes
12	overcast	mild	high	true	yes
13	overcast	hot	normal	false	yes
14	rainy	mild	high	true	no

The decision tree in → Figure 5.5 illustrates the procedure: *Outlook* was selected as the top node. This attribute has three values, so three edges are created: *sunny*, *overcast*, and *rainy*. Now, for each value, those data records from the original training set are selected that have exactly this value as the attribute value for outlook: *sunny*, *overcast*, or *rainy*. For *overcast*, we can immediately enter *yes*, since *play=yes* in all data records where the attribute value is *overcast*. For the other two cases, the process must be continued with the respective subsets of the training set.

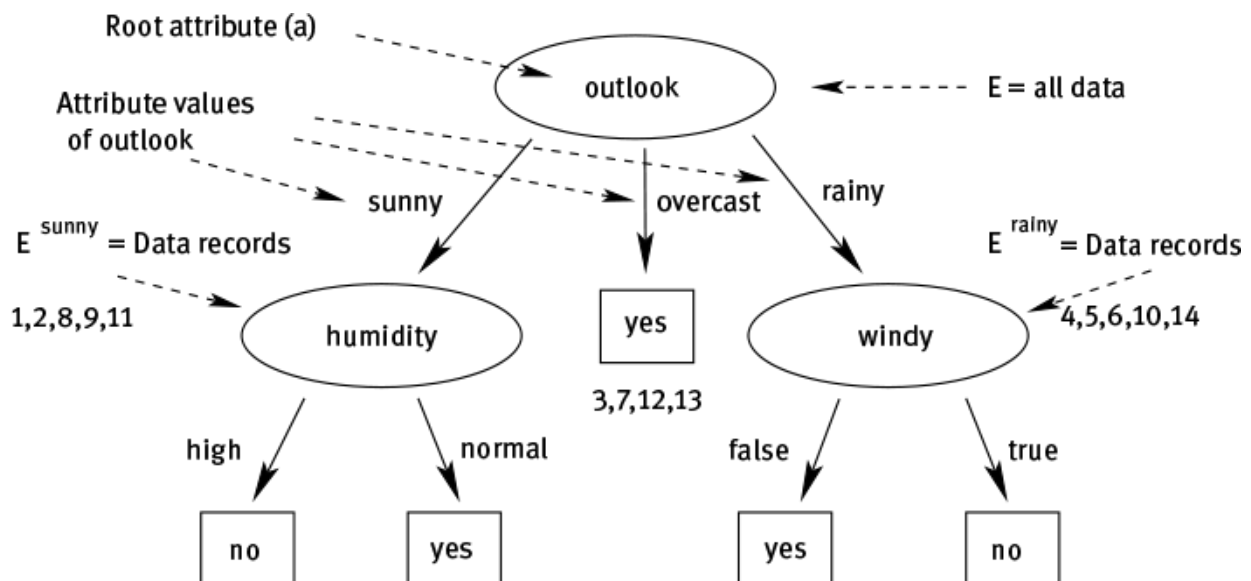


Fig. 5.5 Decision tree for the weather example

This procedure is referred to as *Top down induction of decision trees* (TDIDT method).

The methods discussed in this Section generate a *univariate* tree: exactly one attribute is queried at each node. There are methods that generate *multivariate* decision trees. In these trees, multiple

attributes can be used in a node, for example as a linear combination of attributes:

$$\text{weight} + 2 \cdot \text{size} < 70$$

Univariate trees decompose the multidimensional feature space with axis-parallel cuts, while multivariate trees with a linear combination of attributes still generate linear but not axis-parallel cuts. Non-linear expressions can also be queried:

$$\frac{\text{weight}}{\text{size} \cdot \text{size}} < 25$$

Now the cuts in the feature space are no longer linear.

Multivariate trees have the advantage of usually being more accurate and smaller. However, the disadvantage of them being more complicated to construct outweighs this advantage. In addition, multivariate trees are more difficult to interpret.

Query combinations of attribute values such as *outlook=sunny AND windy=false* can be implemented by arranging nodes in sequence in the tree.

5.2.2 Selection of an attribute

A decision tree is characterized by its nodes, i.e., by the arrangement of attributes in the tree. However, the attribute nodes can be arranged in virtually any order. Consequently, there is no single, correct decision tree for solving a classification problem, but rather a set of possible decision trees. What kind of tree are we aiming for? Of course, we want to generate a tree that is as compact as possible, i.e., one that provides us with a result in the form of a class assignment after just a few questions. However, considering the effort involved, it is impossible to generate and compare all these trees when there are a large number of attributes.

So how do you find suitable attributes that quickly lead to a prediction? There are three ways to select a suitable (next) attribute for the decision tree.

1. **Manual attribute selection**

The attribute to be selected is specified manually by the user. This can only be a sensible solution for small sets of attributes.

2. **Random attribute selection**

An alternative to manual attribute selection is the random generator. However, this can result in trees with very long paths that are confusing and therefore difficult to interpret.

3. **Calculation of attributes**

With this form of attribute selection, a search is automatically performed for an attribute that generates a tree with a small scope. An algorithm selects an attribute and guarantees that a tree with a small scope is created.

Option 3 is our preferred option. To implement it, we need a criterion by which the attributes can be compared with each other and a procedure that selects the optimal attribute.

Example 5.4

(Attribute selection). To select a suitable attribute, we can use, for example, the criterion that it causes the maximum improvement in classification performance *locally*. To do this, each attribute a is tentatively assumed to be the root. The sample set is divided based on the values ω_a of attribute a . For each of these subsets, the most frequent class in the subset is now predicted. Now we determine the error rate for each value $x \in \omega_a$:

$$\text{error}(x) = \frac{\text{incorrectly classified examples}}{\text{all examples}}$$

“All examples” includes all data for which the attribute a has the value x . This number is denoted by $|A_x|$, where $|A|$ is the number of all data records.

The error rate of the attribute a is calculated as the sum of the error rates of its values weighted by their relative frequency.

$$\text{error}_a = \sum_{x \in \omega_a} \left(\frac{|A_x|}{|A|} \cdot \text{error}(x) \right) \rightarrow \min \quad 5.5$$

The attribute with the *lowest error rate* is now selected as the root of the decision tree.

We take an attribute and build the decision tree with this attribute as the root attribute. However, the procedure must already perform a classification at the next level. In → Figure 5.5, we force the procedure to make a classification decision in the branch *outlook=sunny*, which results in 3:2 for *no*. Thus, 2 error points are generated in this case. For *outlook=overcast*, the prediction would be *yes* (4:0, no error points). For *outlook=rainy*, the procedure would predict *yes* (3:2), thus generating 2 more errors points.

We do the same for the other attributes. For the weather example (see → Table 5.6), the following local classification performances, i.e., error rates, result for the attributes *outlook*, *temperature*, *humidity*, and *windy*:

$$\begin{aligned} \text{error}(\text{outlook}) &= \frac{4}{14} \\ \text{error}(\text{temperature}) &= \frac{5}{14} \\ \text{error}(\text{humidity}) &= \frac{4}{14} \\ \text{error}(\text{windy}) &= \frac{5}{14} \end{aligned}$$

For *temperature=hot*, there is a tie, as there are two instances of *play* and two instances of *no play*. But no matter what we predict, we get two error cases.

We can choose between *outlook* and *humidity* based on the local error rates.

The principle of locally best classification has not become widely accepted, as the resulting decision trees usually only achieve moderate quality, i.e., they do not provide a good classification.

Metric attributes

Unfortunately, the concept of the decision tree cannot be easily applied to metric values. With metric attributes, especially those with large value ranges, it is not only inappropriate but also impossible to create a single edge for each value, for example, with real-valued attributes such as annual salary or height. There are two ways of discretizing metric attributes to solve this problem:

Grouping The values are grouped into intervals, see → Chapter 8 and → Section 4.4. An edge is now created for each of the intervals.

Threshold values Exactly two edges are created, which differ in that the attributes fall below or exceed a certain numerical value. With this approach, it is possible to examine an

attribute multiple times with different threshold values within a path of the tree.

5.2.3 The ID3 algorithm for generating a decision tree

Let's return to our preferred method for finding a suitable attribute: automatic generation. The data shown in → Table 5.7 is available for assessing credit risk. The target attribute here is *credit risk*. The aim is to predict the risk based on the other given attributes.

Tab. 5.7 Credit risk

No.	Credit history	Indebtedness	Collaterals	Income	Credit risk
1	poor	high	none	0-15	high
2	unknown	high	none	15-35	high
3	unknown	low	none	15-35	medium
4	unknown	low	none	0-15	high
5	unknown	low	none	> 35	low
6	unknown	low	adequate	> 35	low
7	poor	low	none	0-15	high
8	poor	low	adequate	> 35	medium
9	good	low	none	> 35	low
10	good	high	adequate	> 35	low
11	good	high	none	0-15	high
12	good	high	none	15-35	medium
13	good	high	none	> 35	low
14	poor	high	none	15-35	high

How can we generate a decision tree from this data that helps us decide whether the risk of granting a loan is high, medium, or low from the bank's perspective? The decision tree must also be able to classify cases that do not appear in the table above.

→ Figure 5.6 shows a decision tree that covers all the examples in → Table 5.7. A maximum of four questions must be asked before a prediction can be made.

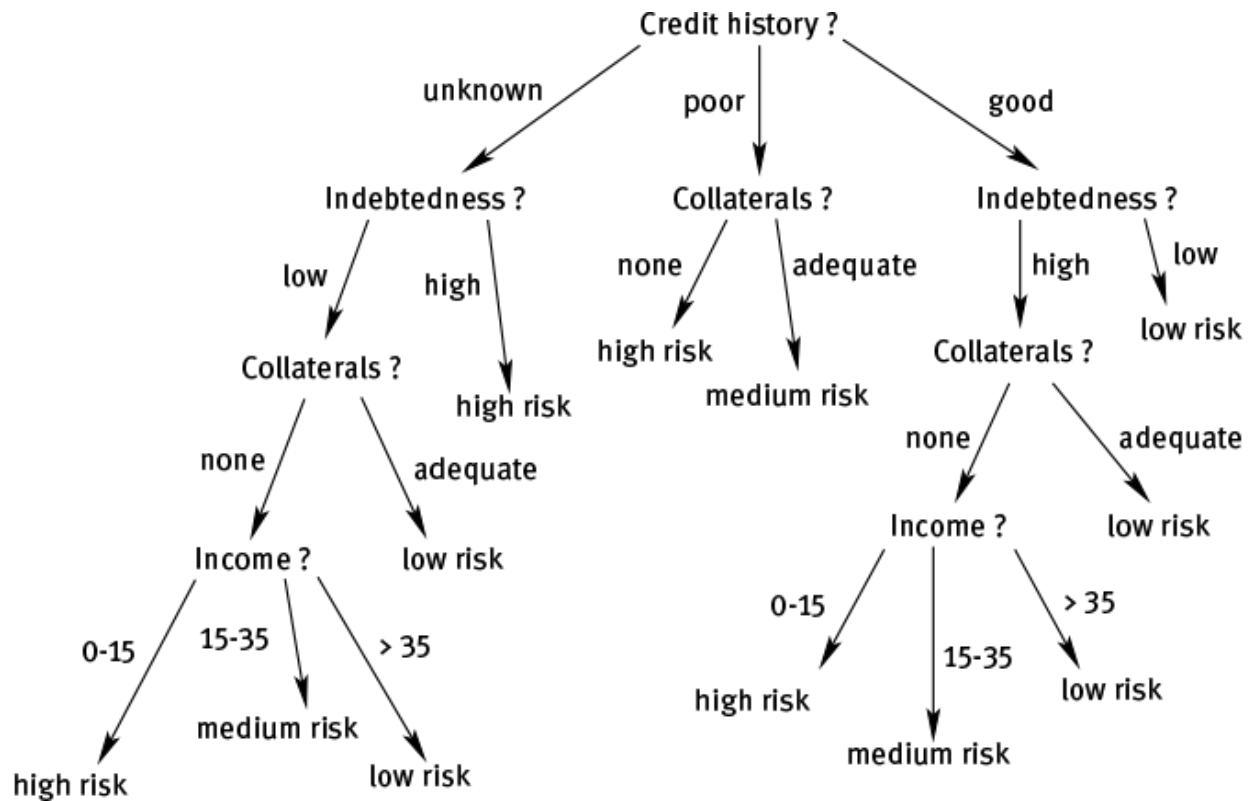


Fig. 5.6 Decision tree 1 for the credit example

The size of the decision tree almost always depends on the order in which the characteristics (attributes) are used for case differentiation. → Figure 5.7 shows a smaller decision tree. In this tree, a maximum of only three questions are required to make a prediction.

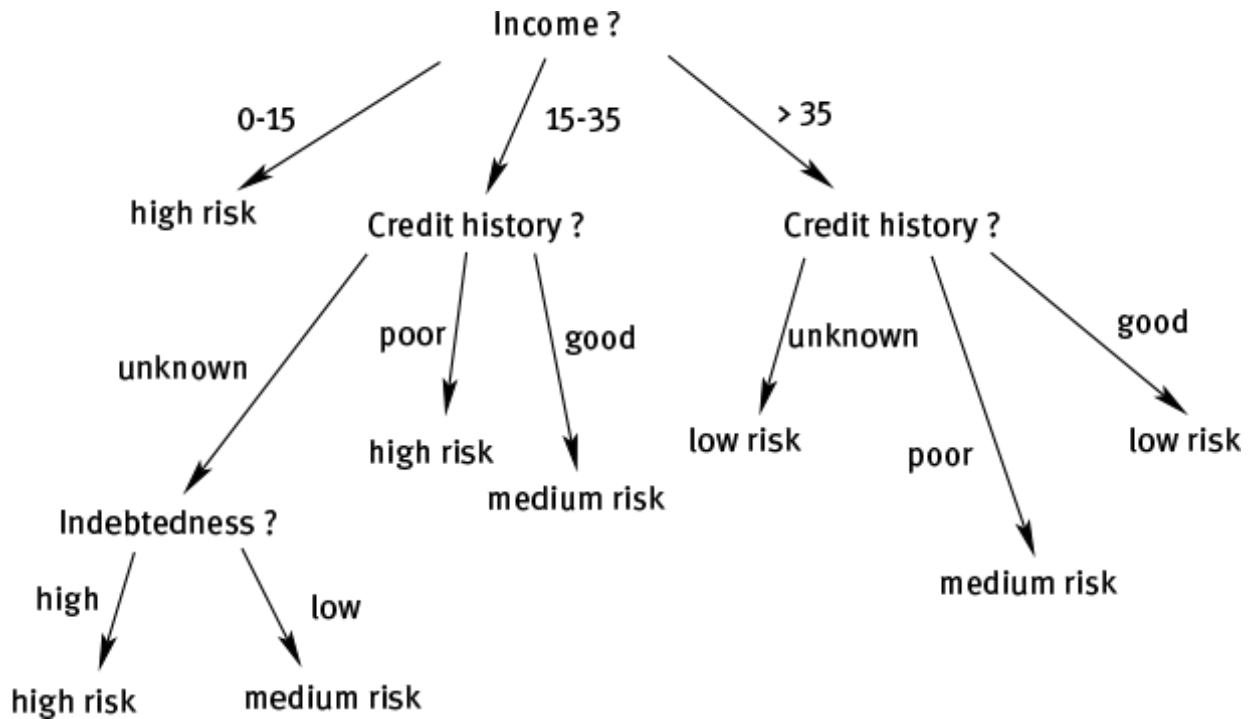


Fig. 5.7 Decision tree 2 for the credit example

Which tree is optimal for classifying the unknown data records? The ID3 algorithm assumes that this is the *simplest* tree (see → Section 9.1). The aim is to create a tree that is as compact and small as possible.

```

FUNCTION induce_tree(example set Ex, attributes Attr)
  IF all entries from Ex belong to the same class
    THEN RETURN leaf node with label of this class
  ELSE
    select an attribute A from Attr
    set A as root for the current tree
    delete A from Attr
    FOREACH value AV of A
      create edge in tree with edge label AV
      let Ex_AV be all elements of example set Ex,
        that currently have AV as the value for A
      result of edge AV := induce_tree(Ex_AV, Attr)
    END FOREACH
  END IF
END.
  
```

Listing 5.6 (ID3 decision tree learning).

The algorithm (→ Listing 5.6) does not specify which attribute should be selected. How does the ID3 algorithm select a suitable attribute? The method presented here was developed in 1979 by Quinlan [→ Qui86] and uses Shannon's concept of *entropy* [→ Sha48]. It is based on information theory.

The attribute that provides the highest *information gain* is selected.

First, let's define the concept of the *entropy* of an attribute, on which the *information gain* is based.

Definition 5.7

(Entropy). The *entropy* of an attribute B is measured as

$$I(B) = \sum_{i=1}^k -\rho(b_i) \cdot \log_2(\rho(b_i)) \quad 5.6$$

where

- the b_i are the possible values of attribute B ,
- k is the number of values of attribute B , and
- ρ is the probability (more precisely: relative frequency) of b_i occurring in the training set.

If $p(b_i) = 0$, we set the term $-p(b_i) \cdot \log_2(p(b_i))$ to 0.

The entropy always refers to a single attribute and a set of given data records. The above definition refers to the relative frequency of a value b_i of the attribute B . To do this, the reference set must be defined. Since the entropy always refers to the target attribute, we omit the target attribute and write $I(\text{table})$ instead of $I_{\text{target attribute}}(\text{table})$ for simplicity.

The entropy can be interpreted as the impurity or disorder of a set. The higher the entropy, the greater the disorder, and vice versa. Let us consider an example set with two classes. If only one class occurs in a subset, then according to the definition of entropy, the value for this subset is 0. This corresponds to our idea, because this subset contains no disorder with regard to the target attribute, since all data in this subset belong to the same class.

Example 5.5

(Entropy). How high is the entropy or impurity of the data in → Table 5.7 (p. 96)? To answer this question, we first need the relative frequencies of the target attribute values.

- $p(\text{high risk}) = \frac{6}{14}$
- $p(\text{medium risk}) = \frac{3}{14}$
- $p(\text{low risk}) = \frac{5}{14}$

Consequently, the entropy of the entire table is

$$I(\text{Tab}) = I_{\text{risk}}(\text{Tab}) = -\frac{6}{14} \cdot \log_2\left(\frac{6}{14}\right) - \frac{3}{14} \cdot \log_2\left(\frac{3}{14}\right) - \frac{5}{14} \cdot \log_2\left(\frac{5}{14}\right) = 1.531$$

Now we select the attribute that achieves the *maximum information gain* or provides the maximum reduction in impurity. To do this, we calculate the difference between $I(\text{table})$ and the impurity that results for the next level, see → Figure 5.8.

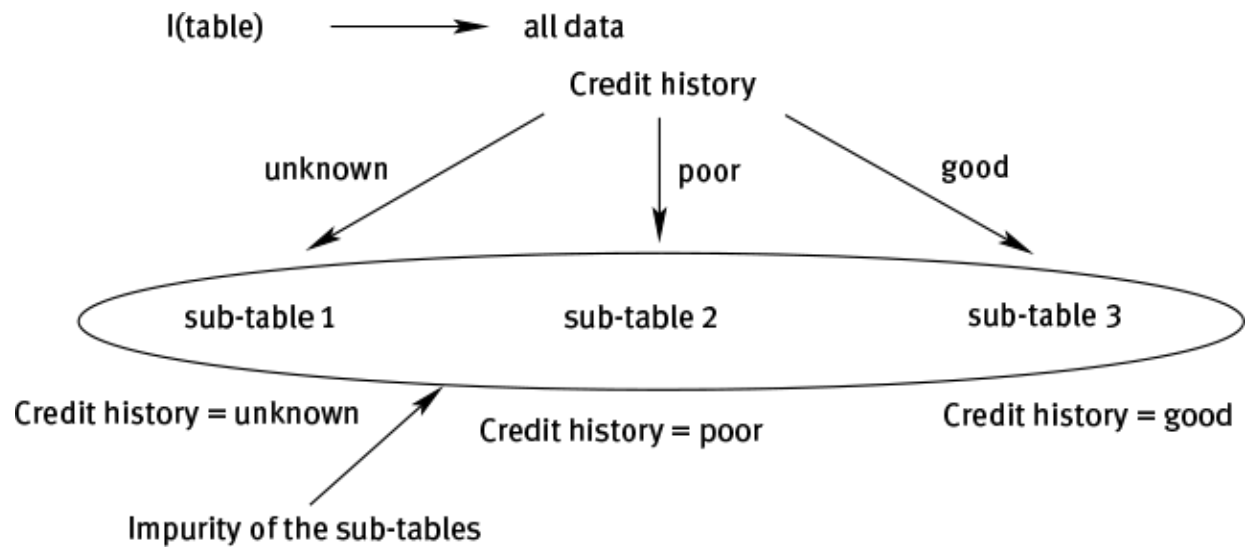


Fig. 5.8 Impurity when using the attribute *credit history*

Let E be an example set, the complete training set. If, for example, attribute B with n values is selected, E is divided into n subsets (sub-tables): $\{E_1, \dots, E_n\}$.

After B has been set as the root of the tree, the remaining impurity of the decision tree is calculated from the weighted sum of the entropies $I(E_j)$:

$$G(B) = \sum_{j=1}^n \frac{|E_j|}{|E|} \cdot I(E_j) \quad 5.7$$

G is the sum of the entropies of the sub-trees resulting from B , weighted according to their relative frequency.

The *information gain* is then calculated as

$$\text{gain}(B) = I(E) - G(B) \quad 5.8$$

The aim is to maximize the *gain*. To do this, the above calculation is performed for all attributes. The attribute B that yields the maximum $\text{gain}(B)$ is selected. For our example, we first select *credit history* as the attribute. This attribute has three values: *unknown*, *poor*, *good*. For each of these values, we count how often they are associated with which risk in the data records:

Value	High risk	Medium risk	Low risk
unknown	2	1	2
poor	3	1	0
good	1	1	3

The result is:

- $I(\text{credit history\_unknown}) = -\frac{2}{5} \cdot \log_2\left(\frac{2}{5}\right) - \frac{1}{5} \cdot \log_2\left(\frac{1}{5}\right) - \frac{2}{5} \cdot \log_2\left(\frac{2}{5}\right) = 1.52$
- $I(\text{credit history\_poor}) = -\frac{3}{4} \cdot \log_2\left(\frac{3}{4}\right) - \frac{1}{4} \cdot \log_2\left(\frac{1}{4}\right) = 0.81$
- $I(\text{credit history\_good}) = -\frac{1}{5} \cdot \log_2\left(\frac{1}{5}\right) - \frac{1}{5} \cdot \log_2\left(\frac{1}{5}\right) - \frac{3}{5} \cdot \log_2\left(\frac{3}{5}\right) = 1.37$

Thus, when using the *credit history* attribute, we obtain

$$G(\text{credit history}) = \sum_{j=1}^n \frac{|E_j|}{|E|} I(E_j) = \frac{5}{14} \cdot 1.52 + \frac{4}{14} \cdot 0.8113 + \frac{5}{14} \cdot 1.37095 = 1.265$$

→ Figure 5.9 illustrates the procedure.

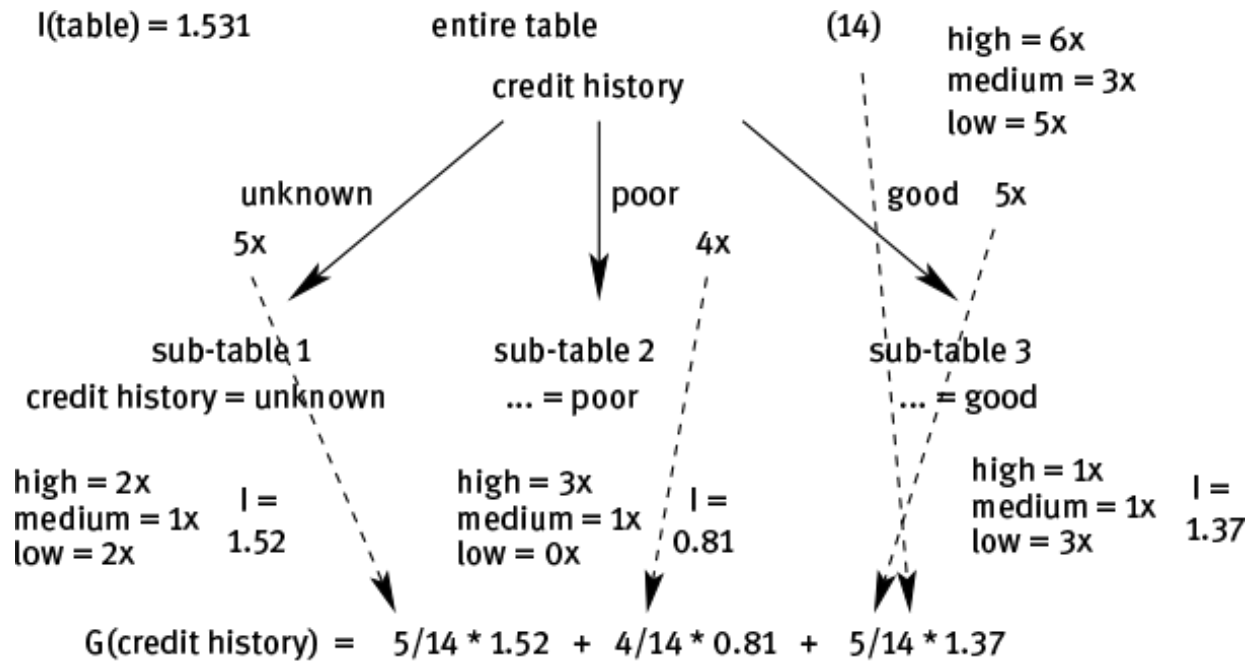


Fig. 5.9 G calculation

We perform this calculation for all four attributes and obtain:

gain(credit history)	= 1.531 - 1.265	= 0.266
gain(income)	= 1.531 - 0.564	= 0.967
gain(indebtedness)	= 1.531 - 1.468	= 0.063
gain(collaterals)	= 1.531 - 1.325	= 0.206

The gain for the attribute *income* is greatest, so this attribute is selected as the root node. The process is then continued recursively: a tree is generated again for each subset assigned to a node. We generate the sub-tree for the branch *income*=15-35. It is important to note that we no longer need to consider the entire table, but only the data records where *income*=15-35. We can therefore delete a number of data records in → Table 5.7 and obtain → Table 5.8. The *income* column can be deleted completely, as this characteristic has already been processed.

Tab. 5.8 Data for assessing credit risk, revised version

No.	Credit history	Indebtedness	Collaterals	Income	Credit risk
2	unknown	high	none	15-35	high
3	unknown	low	none	15-35	medium
12	good	high	none	15-35	medium
14	poor	high	none	15-35	high

Which attribute do we choose next for the branch *income=15-35*? To find this out, we first calculate the entropy of → Table 5.8.

- $p(\text{high risk}) = \frac{2}{4}$
- $p(\text{medium risk}) = \frac{2}{4}$
- $p(\text{low risk}) = \frac{0}{4}$

Consequently: $I(\text{table}) = -\frac{2}{4} \cdot \log_2\left(\frac{2}{4}\right) - \frac{2}{4} \cdot \log_2\left(\frac{2}{4}\right) - \frac{0}{4} \cdot \log_2\left(\frac{0}{4}\right) = 1$

We look for the attribute that achieves the maximum information gain. We recalculate the difference between $I(\text{table})$ and the impurity that results for the next level.

First, let's select *credit history* as our attribute. There are three possible values for credit history: *unknown*, *poor*, and *good*. For each of these values, we count how often each risk occurs:

Value	High risk	Medium risk	Low risk
unknown	1	1	0
poor	1	0	0
good	0	1	0

The result is

$$I(\text{credit history\_unknown}) = -\frac{1}{2} \cdot \log_2\left(\frac{1}{2}\right) - \frac{1}{2} \cdot \log_2\left(\frac{1}{2}\right) - \frac{0}{2} \cdot \log_2\left(\frac{0}{2}\right) = 1$$

$$I(\text{credit history\_poor}) = -\frac{1}{1} \cdot \log_2\left(\frac{1}{1}\right) - \frac{0}{1} \cdot \log_2\left(\frac{0}{1}\right) - \frac{0}{1} \cdot \log_2\left(\frac{0}{1}\right) = 0$$

$$I(\text{credit history\_good}) = -\frac{0}{1} \cdot \log_2\left(\frac{0}{1}\right) - \frac{1}{1} \cdot \log_2\left(\frac{1}{1}\right) - \frac{0}{1} \cdot \log_2\left(\frac{0}{1}\right) = 0$$

This yields

$$G(\text{credit history}) = \sum_{j=1}^n = \frac{|E_j|}{|E|} I(E_j) = \frac{2}{4} \cdot 1 + \frac{1}{4} \cdot 0 + \frac{1}{4} \cdot 0 = 0.5$$

We perform this calculation for the other attributes as well and obtain:

- $\text{gain}(\text{credit history}) = 1 - 0.5 = 0.5$
- $\text{gain}(\text{indebtedness}) = 1 - 0.6887 = 0.3113$
- $\text{gain}(\text{collaterals}) = 1 - 1 = 0$

The next node selected is *credit history*, because this is where the gain is greatest.

→ Figures 5.10 and → 5.11 show the KNIME workflow and the decision tree generated by WEKA (in KNIME). Instead of the ID3 algorithm, we chose the J48 algorithm, a variant of C4.5 (→ Section 5.2.6),

because this algorithm provides a graphical representation of the decision tree.

CSV Reader

J48 (3.7)

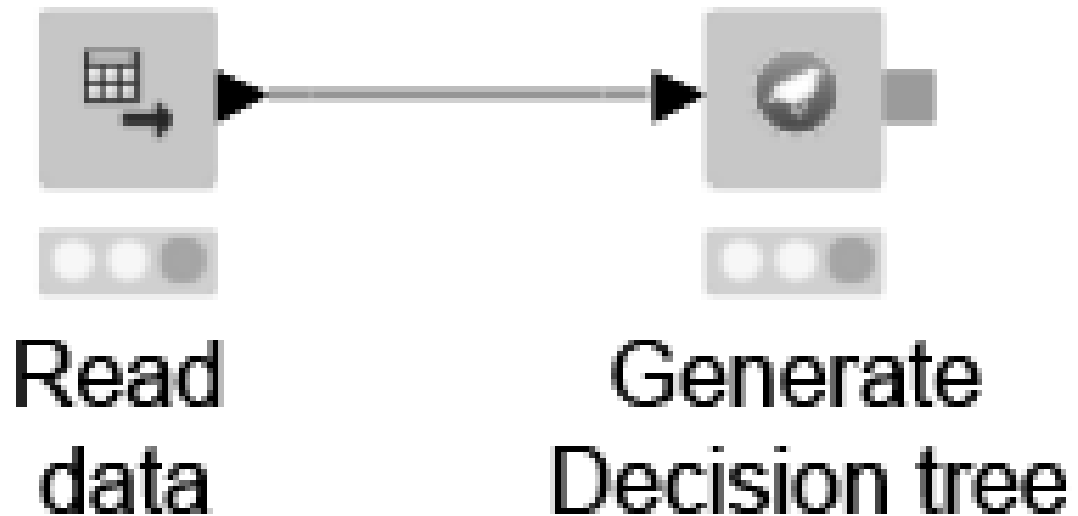


Fig. 5.10 Workflow for assessing creditworthiness

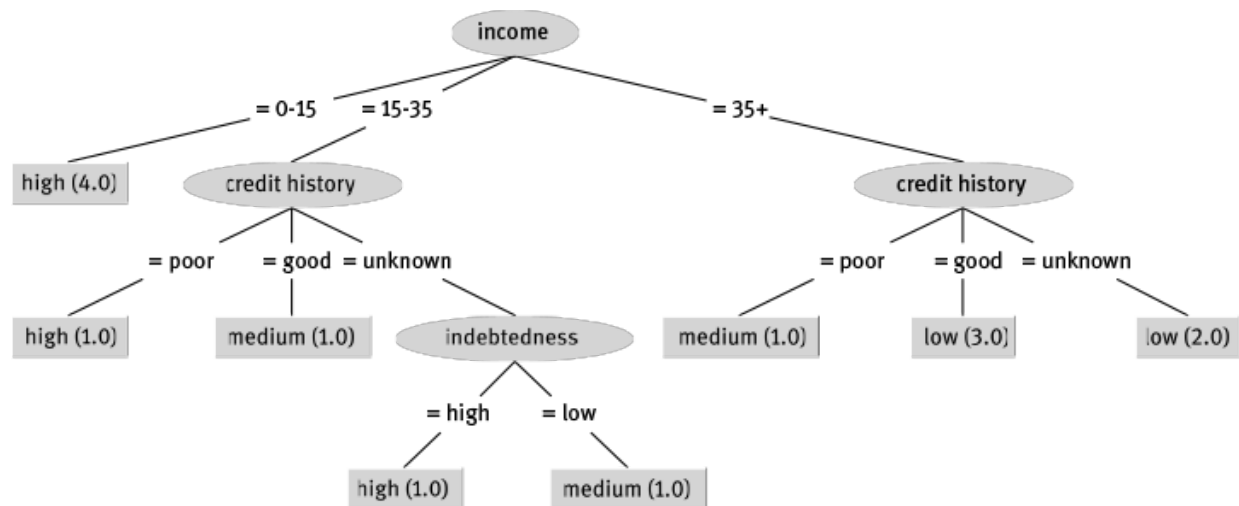


Fig. 5.11 Decision tree for the problem of creditworthiness

5.2.4 Entropy

Let's return to the concept of *entropy*. So far, we have simply used the formula for entropy in → Definition 5.7. To gain a better understanding, it is helpful to look at the background of this formula.

5.2.4.1 Variant 1

Let E be an event that occurs with probability $p(E) > 0$. We say that the message that the event E has occurred contains

$$I(E) = \log_2 \frac{1}{p(E)}$$

units of information. If $p(E) = 1$, then we are dealing with a certain event. In this case, there is no information contained within it: the event will definitely (always) occur. $I(E) = 0$ applies, which correctly reflects this. When do we get $I(E) = 1$? This is when $p(E) = \frac{1}{2}$, i.e., when we have two equally probable events. The less probable the event, the greater $I(E)$ becomes. This also corresponds to our intuition: *the less probable an event, the greater the information if it occurs.*

Now let us consider a random experiment: k events $a_1, a_2, a_3, \dots, a_k$ are possible, with the probabilities $p_1, p_2, p_3, \dots, p_k$. We assume that

$$\sum_{i=1}^k p_i = 1, \quad p_i \geq 0$$

If the event a_i occurs, then we obtain $I(a_i) = \log_2 \frac{1}{p_i}$ bits of information.

The event occurs with a probability of p_i . The average amount of information obtained is therefore

$$\sum_{i=1}^k p_i \cdot I(a_i) = \sum_{i=1}^k p_i \cdot \log_2 \frac{1}{p_i}$$

On this basis, we can now define *entropy*.

definition 5.8

definition 5.8 (Entropy)

Entropy is defined as

$$H_k := H_k(p_1, p_2, \dots, p_k) = - \sum_{i=1}^k p_i \cdot \log_2(p_i)$$

In the case of $p_i = 0$, we set the term $p_i \cdot \log_2(p_i)$ to 0.

We can interpret $I(a_i)$ as the information needed to specify that a_i occurs. H_k can be understood as the average amount of information per outcome of the random experiment. However, H can also be interpreted as the average uncertainty that an observer has *before* the experiment.

5.2.4.2 Variant 2

The second variant of the derivation comes from cryptography and develops the concept of entropy on the basis of coding, cf. [→ PM23, → Section 7.2].

A bit is a binary number. It can represent two pieces of information, i.e., distinguish between two things. With two bits, four things can be encoded (00, 01, 10, 11), with three bits eight things, etc. In general, with k bits, 2^k cases can be distinguished. Conversely, k different things can be represented with $\log_2(k)$ bits.

However, if we consider the probabilities of an event occurring, we can improve this even further. We want to distinguish between the elements of the set $\{a, b, c, d, e\}$ and consider the following binary coding for this purpose:

a	0	b	10	c	110	d	1110	e	1111

Assume that the probabilities for the occurrence of the elements are:

$$p(a) = \frac{1}{2}, p(b) = \frac{1}{4}, p(c) = \frac{1}{8}, p(d) = \frac{1}{16}, p(e) = \frac{1}{16}$$

So this encoding sometimes uses one bit (a), sometimes two (b), three (c) or even four (d and e) bits. To determine the average number of bits used with this encoding, we must take into account the probabilities of the respective elements occurring:

$$P(a) \cdot 1 + p(b) \cdot 2 + p(c) \cdot 3 + p(d) \cdot 4 + p(e) \cdot 4 = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{16} \cdot 4 + \frac{1}{16} \cdot 4$$

And this is, of course, better than encoding using three bits, which would be required for five elements according to the rounded logarithm $\log_2(5)$.

With this encoding, we therefore need one bit to distinguish a from the other elements. That is $-\log_2(p(a)) = 1$. Four bits are used to distinguish d and e from the other elements. That is $-\log_2(p(d)) = 4$.

We now construct a coding scheme that uses exactly $-\log_2(p(x))$ bits (or the next largest natural number) to encode x . And now it is clear that information transmission requires

$$\sum_{i=1}^k -p_i \cdot \log_2(p_i)$$

bits on average. This is just the entropy.

Huffman coding, which is used in cryptography, is based on this approach.

5.2.5 The Gini index

Alternatively, the so-called *Gini index* can be used instead of the entropy and gain. The Gini index was developed by the Italian statistician Corrado Gini (1884-1965).

The *Gini index* is the equivalent of the entropy of a table with respect to a target attribute B ; it is given by

$$\text{gini}_B(E) = 1 - \sum_{i=1}^k p(b_i)^2 \quad 5.9$$

Here, the b_i represent the possible values of attribute B , and p is the probability (the relative frequency) of b_i occurring.

Similar to gain, we then define

$$\text{GINI}(B) = \sum_{j=1}^n \frac{|E_j|}{|E|} \cdot \text{gini}_B(E_j) \quad 5.10$$

The GINI is chosen for selecting the best attribute.

The gain is now calculated as follows:

$$\text{gain}(B) = \text{gain}(E) - \text{GINI}(B) \quad 5.11$$

The selected attribute B is again the one that provides the maximum improvement, i.e., the maximum gain.

5.2.6 The C4.5 algorithm

One disadvantage of the ID3 algorithm is that it only works with nominal or ordinal attributes and cannot handle numerical attributes. The successor to ID3, the C4.5 algorithm [Qui93], can do this. The C4.5 algorithm divides numerical attributes into *intervals* and converts them into ordinal attributes. If an attribute A can take on n values $A_1, \dots, A_n$ then for each $i = 1 \dots n - 1$ the intervals $[a | a \leq A_i]$ and $[a | a > A_i]$ are formed. These two intervals are considered new values of attribute A . The interval formation that yields the greatest gain is selected.

The ID3 algorithm has another disadvantage: the sorting of attributes (as shown above) favors attributes with many different characteristics, attributes with many different values. That is why C4.5 normalizes the information gain. Let B be the attribute whose gain we want to calculate. Let

$$\text{SplitInfo}(B) = - \sum_{j=1}^n \frac{|E_j|}{|E|} \cdot \log_2 \left(\frac{|E_j|}{|E|} \right) \quad 5.12$$

The *information gain* is then normalized:

$$\text{gain}'(B) = \frac{\text{gain}(B)}{\text{SplitInfo}(B)} \quad 5.13$$

Example 5.6

(Split info). Let us consider an excerpt from a table containing data on movie theater visits. We will focus on just two attributes: the *ticket price* and the target attribute *movie theater visited*.

price	6	11	7	6	10	10	6	7	11	10	11	10
visited	y	n	y	y	n	n	y	n	n	y	n	n

We consider two conversions of the price attribute into ordinal attributes. Variant 1:

- 6 and 7 become *cheap*: 5 items.
- 10 and 11 become *expensive*: 7 items.

Variant 2:

- 6 becomes *cheap*: 3 items.
- 7 becomes *moderate*: 2 items.
- 10 becomes *expensive*: 4 items.
- 11 becomes *very expensive*: 3 items.

For variant 1, this results in

$$G = -\frac{5}{12} \cdot \left(\frac{4}{5} \cdot \log_2\left(\frac{4}{5}\right) - \frac{1}{5} \cdot \log_2\left(\frac{1}{5}\right) \right) - \frac{7}{12} \cdot \left(\frac{1}{7} \cdot \log_2\left(\frac{1}{7}\right) - \frac{6}{7} \cdot \log_2\left(\frac{6}{7}\right) \right) = 0.65$$

For variant 2, the value is 0.44. This clearly shows that the gain is greater for variant 2: The entropy of the entire table is $1(E) = 0.98$. So the gain for variant 2 is 0.54, while for variant 1 it is only 0.33.

This is a typical effect; attributes with many values are preferred. To compensate for this, the information gain is divided by the SplitInfo value.

For variant 1, we get a SplitInfo value of

$$\text{SplitInfo}_1 = -\frac{5}{12} \cdot \log_2\left(\frac{5}{12}\right) - \frac{7}{12} \cdot \log_2\left(\frac{7}{12}\right) = 0.98$$

For variant 2, the result is: 1.96. The larger SplitInfo value reduces the gain for variant 2 more than the gain for variant 1, thereby reducing the preference effect.

With SplitInfo, the gain for variant 1 is $\frac{0.33}{0.98} = 0.34$, and for variant 2 it is only $\frac{0.54}{1.96} = 0.28$.

Since the C4.5 method can handle numerical values, it is now possible to generate a decision tree for the Iris data (see → Appendix A.1). → Figure 5.12 shows the decision tree generated by WEKA, which was calculated using the J48 algorithm, WEKA's variant of C4.5.

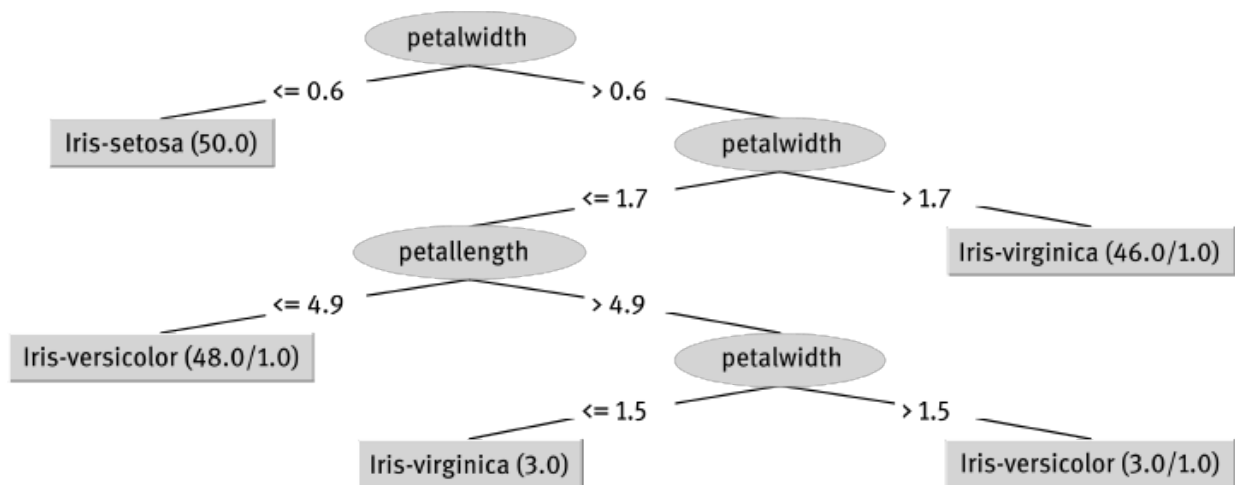


Fig. 5.12 Decision tree for Iris data

5.2.7 Problems with decision tree learning

When generating a decision tree, it can happen that the decision tree memorizes the training data. It will then classify *all* training data correctly. However, such decision trees often do not work well with test data, since they only *memorized* all the training data. This effect, known as *overfitting*, can be prevented by artificially shortening the tree. There are two simple approaches:

1. Each subtree must contain a minimum number of data records. No further subtrees are generated if the number of data records in the training set falls below a certain threshold.
2. Initially, the complete decision tree is generated, then some generated subtrees are replaced by leaf nodes.

Both approaches belong to the so-called *pruning* techniques, of which there are several others. For example, it can be required that the improvement in entropy (i.e., disorder) must not be too small.

With pruning, we give up the desire for our decision tree to predict the training data 100 % correctly. This can happen anyway if we have contradictory data. For example, there may be two customers with identical profiles. However, one customer is a good customer and the other is a poor customer. In this case, our decision tree cannot work 100 % correctly on the training data.

A decision tree that has been pruned can achieve significantly better results in later applications than a decision tree that works completely correctly on all training data. By pruning, we force the resulting decision tree to generalize.

Decision trees are very popular. This is partly because they are easy to interpret. Another reason is that decision tree methods are not complicated and therefore easy to implement. One disadvantage of decision tree methods is that they cannot be developed incrementally. Once a decision tree has been generated, it cannot simply be modified if new data is added. Instead, a new decision tree must be generated.

In addition to the methods presented here, there are other decision tree methods, such as NewID, ID5R, CHAID, CLS, LCLRD, and AQ15.

Generating a good decision tree often depends on selecting a good training set, i.e., the subset of the given data used to generate the decision tree.

We can address this problem with an approach that represents a kind of democratic procedure. We generate several decision trees and simply make majority decisions when it comes to predictions. This approach is called *random forest* and belongs to the category of *ensemble learning*. A series of different procedures is carried out and the results are combined to derive one decision, see → Section 5.6.

In this section, we have focused on classification trees. These belong to the TDIDT (*top-down induction of decision trees*) methods, a class of methods that also includes regression trees. Regression trees work on metric data. One example is the CART algorithm, which generates a binary tree. The basic principle is comparable to the ID3 and C4.5 methods discussed in this section, which aim to bring order to our data quickly. CART uses a different measure of disorder: the Gini impurity. For more details, see [→ Fro21].

5.2.8 Decision tree and rules

If we want to use our data mining model for decision support, we need a machine-readable representation. The decision tree is unsuitable for this in its graphical representation. However, we can easily generate *rules* (see → Section 4.3) from a decision tree. From the decision tree shown in → Figure 5.11, rules representing the entire tree can be easily extracted, for example:

income = > 35 AND creditworthiness = unknown THEN risk = low income = 6

There are a number of methods for generating rules that are not based on decision trees. In addition to the methods presented in → Chapter 7 on association analyses, which can also be easily applied to classification tasks, there are other methods such as AQ15, CN2, and LCLR [→ Pet05].

Task 5.4

Task 5.4 (Decision tree, weather example)

Let us consider the weather example again, but now with the numerical values for *temperature* and *humidity*, see → Appendix A.3. Determine a decision tree for the attribute *play*, which indicates

whether the game will be played under the given weather conditions. How do you handle the numerical values?

Outlook	Temp (°F)	Humidity (%)	Windy	Play
sunny	85	85	false	no
sunny	80	90	true	no
overcast	83	78	false	yes
rainy	70	96	false	yes
rainy	68	80	false	yes
rainy	65	70	true	no
overcast	64	65	true	yes
sunny	72	95	false	no
sunny	69	70	false	yes
rainy	75	80	false	yes
sunny	75	70	true	yes
overcast	72	90	true	yes
overcast	81	75	false	yes
rainy	71	80	true	no

Task 5.5

Task 5.5 (Decision tree, skiing example). Use ID3 to generate a decision tree to predict whether skiing will take place.

The table contains the following attributes:

- Dist: Distance to ski resort (more/less than 100 km)
- Sunny: (yes/no)
- Weekend: Is it a weekend? (yes/no)
- Skiing (**target attribute**): I will go skiing. (yes/no)

Day	Dist	Weekend	Sunny	Skiing
1	l	y	y	y
2	l	y	y	y
3	l	y	n	y
4	l	n	y	y
5	m	y	y	y
6	m	y	y	y
7	m	y	y	n
8	m	y	n	n
9	m	n	y	n
10	m	n	y	n
11	m	n	n	n

Task 5.6

Task 5.6 (Decision tree, contact lens example).

Find a decision tree for the contact lens example explained in → Appendix A.4.

Task 5.7

Task 5.7 (Decision tree, restaurant data).

→ Table 5.9 (see → Task 5.1) contains 12 data records with the following attributes:

- Alternative: Is there another suitable restaurant nearby? {yes, no}
- Fri/Sat: Is it Friday or Saturday? {yes, no}
- Hungry: Am I hungry? {yes, no}
- Guests: How many people are in the restaurant? {none, a few, full}
- Reservation: Have I made a reservation? {yes, no}
- Type: What kind of restaurant is it? {French, Chinese, Italian, Burger}
- Waiting time: How long does the restaurant estimate the wait will be? {0-10, 10-30, 30-60, >60}
- Waiting (target attribute): Am I waiting? {yes, no}

Generate a decision tree and classify the last 3 data records in the table.

Tab. 5.9 Restaurant example data

Alternative	Fri/Sat	Hungry	Guests	Reservation	Type	Waiting time	Waiting
yes	no	yes	a few	yes	French	0-10	yes
yes	no	yes	full	no	Chinese	30-60	no
no	no	no	a few	no	Burger	0-10	yes
yes	yes	yes	full	no	Chinese	10-30	yes
yes	yes	no	full	yes	French	>60	no
no	no	yes	a few	yes	Italian	0-10	yes
no	no	no	none	no	Burger	0-10	no
no	no	yes	a few	yes	Chinese	0-10	yes
no	yes	no	full	no	Burger	>60	no
yes	yes	yes	full	yes	Italian	10-30	no
no	no	no	none	no	Chinese	0-10	no
yes	yes	yes	full	no	Burger	30-60	yes
yes	no	yes	a few	no	French	30-60	
yes	yes	yes	full	yes	Chinese	10-30	
no	no	no	none	no	Burger	0-10	

5.3 Naive Bayes

The naive Bayes algorithm is a probability-based method. It aims to predict the *most probable* class. No model training takes place. The prediction is calculated directly from the training data. We start from a basic assumption: All attributes are *independent* of each other.

The naive Bayes algorithm is based on *Bayes' formula*, which allows dependent events to be swapped in calculations with conditional probabilities.

5.3.1 Bayesian formula

Let Y be an event with $P(Y) > 0$. Then

$$P(X|Y) = \frac{P(X \wedge Y)}{P(Y)} \quad 5.14$$

is called the *conditional probability* of X given the condition Y . If $P(Y) = 0$, then the conditional probability is undefined.

The formula for conditional probability can be rearranged to

$$P(X \wedge Y) = P(Y) \cdot P(X|Y)$$

Similarly, the conditional probability of Y with respect to X can be written as:

$$P(Y \wedge X) = P(X) \cdot P(Y|X)$$

Since the logical AND operation is commutative, i.e.

$$P(Y) \cdot P(X|Y) = P(X) \cdot P(Y|X)$$

we obtain the *Bayesian formula*:

$$P(X|Y) = \frac{P(Y|X) \cdot P(X)}{P(Y)} \quad 5.15$$

This formula is very useful for many problems, as it allows the calculation of $P(X|Y)$ from $P(Y|X)$. Often, only one of the two conditional probabilities is available. The Bayesian formula can therefore be used to calculate the inverse conditional probability.

5.3.2 The naive Bayes algorithm

Let's return to the weather example from → Appendix A.3. Let $A = \{a_1, \dots, a_n\}$ be a set of attribute values, for example $\{sunny, hot\}$. We now want to calculate whether the game will be played if it is *sunny* and *hot*. Using Bayes' formula, we obtain

$$P(yes | [sunny, hot]) = \frac{P([sunny, hot] | yes) \cdot P(yes)}{P([sunny, hot])}$$

Similarly, we calculate

$$P(no | [sunny, hot]) = \frac{P([sunny, hot] | no) \cdot P(no)}{P([sunny, hot])}$$

We then predict *yes*, if $P(yes | [sunny, hot])$ is greater than $P(no | [sunny, hot])$, and otherwise *no*.

We see that the term $P([sunny, hot])$ appears in the denominator of both formulas. Since it is the same for both expressions and we are only comparing them in terms of magnitude, we can omit it. However, we are no longer talking about the probability P , but about the so-called *Likelihood* L :

$$L(yes | [sunny, hot]) = P([sunny, hot] | yes) \cdot P(yes)$$

$$L(no | [sunny, hot]) = P([sunny, hot] | no) \cdot P(no)$$

The naive Bayes algorithm assumes that the attributes are independent and replaces:

$$P([sunny, hot]|yes) = P(sunny|yes) \cdot P(hot|yes)$$

The prediction of a class is thus made using the following variables:

1. **Relative frequency** $h[a, k]$ of attribute a in class k
2. **Likelihood** $L[k](A) = \prod_{i=1}^n h[a_i, k] \cdot h[k]$
3. **Probability** $L[k_j](A) = \frac{L[k_j](A)}{\sum_i L[k_i](A)}$
4. **Prediction** $k_m : P[k_m](A) = \max_j (P[k_j](A))$

The relative frequency $h[a, k]$ is exactly our $P(a|k)$. However, we replace $P(a|k)$ with $h[a, k]$, since strictly speaking, the information about the *probability* is not available, but only a small amount of given sample data. We already introduced the *likelihood* above.

We compare the *likelihoods* with each other and take the largest value. To obtain *probability* values again, we normalize our *likelihoods* so that their sum is 1. We achieve this by dividing all likelihoods by the total sum of all likelihoods: $\sum_i L[k_i](A)$

We use the symbol P differently here. In the introductory remarks on probability, we use the symbol P correctly to denote probability. In the actual prediction, however, we would have to assign a different name, because the normalized likelihood is not the probability, but only a kind of probability substitute.

We explain the procedure using the weather example, see → Table 5.10.

Tab. 5.10 Weather data, nominal

Day	Outlook	Temperature	Humidity	Windy	Play
1	sunny	hot	high	false	no
2	sunny	hot	high	true	no
3	overcast	hot	high	false	yes
4	rainy	mild	high	false	yes
5	rainy	cool	normal	false	yes
6	rainy	cool	normal	true	no
7	overcast	cool	normal	true	yes
8	sunny	mild	high	false	no
9	sunny	cool	normal	false	yes
10	rainy	mild	normal	false	yes
11	sunny	mild	normal	true	yes
12	overcast	mild	high	true	yes
13	overcast	hot	normal	false	yes
14	rainy	mild	high	true	no

Will the game be played on a day with the following conditions?

outlook = sunny, humidity = normal, temperature = hot, windy = true

Step 1

First, we determine the *relative frequencies*. The result is $h[\text{sunny}, \text{yes}] = \frac{2}{9}$ because

- there are 9 days on which the game was played ($\text{play}=\text{yes}$), and
- on 2 of these days there was $\text{outlook}=\text{sunny}$.

The other relative frequencies we need are marked in bold in → Table 5.11.

Tab. 5.11 Relative frequencies for weather data

		play	
		yes	no
outlook	sunny	2/9	3/5
	overcast	4/9	0/5
	rainy	3/9	2/5
temperature	hot	2/9	2/5
	mild	4/9	2/5
	cool	3/9	1/5
humidity	high	3/9	4/5
	normal	6/9	1/5
windy	true	3/9	3/5
	false	6/9	2/5

Step 2

Now the *likelihoods* are calculated.

$L[\text{yes}](\text{sunny}, \text{hot}, \text{normal}, \text{true})$

$$\begin{aligned}
 &= h[\text{sunny}, \text{yes}] \cdot h[\text{hot}, \text{yes}] \cdot h[\text{normal}, \text{yes}] \cdot h[\text{true}, \text{yes}] \cdot h[\text{yes}] \\
 &= \frac{2}{9} \cdot \frac{2}{9} \cdot \frac{6}{9} \cdot \frac{3}{9} \cdot \frac{9}{14} \\
 &= 0.007055
 \end{aligned}$$

$L[\text{no}](\text{sunny}, \text{hot}, \text{normal}, \text{true})$

$$\begin{aligned}
 &= h[\text{sunny}, \text{no}] \cdot h[\text{hot}, \text{no}] \cdot h[\text{normal}, \text{no}] \cdot h[\text{true}, \text{no}] \cdot h[\text{no}] \\
 &= \frac{3}{5} \cdot \frac{2}{5} \cdot \frac{1}{5} \cdot \frac{3}{5} \cdot \frac{5}{14} \\
 &= 0.010286
 \end{aligned}$$

Step 3

We calculate the *probabilities*, abbreviating ($\text{sunny}, \text{hot}, \text{normal}, \text{true}$) with A.

$$\begin{aligned}
 P[\text{yes}](A) &= \frac{L[\text{yes}](A)}{L[\text{yes}](A) + L[\text{no}](A)} \\
 &= \frac{0.007055}{0.007055 + 0.010286} = 0.406835 \\
 P[\text{no}](A) &= \frac{L[\text{no}](A)}{L[\text{yes}](A) + L[\text{no}](A)} \\
 &= \frac{0.010286}{0.007055 + 0.010286} = 0.593165
 \end{aligned}$$

Step 4

We predict the class that provides the highest probability:

$$(P[yes](A) = 40.68 \%) < (P[no](A) = 59.32 \%) \Rightarrow NO$$

So, on a sunny, hot, but stormy day with normal humidity, there is a 41 % probability of playing and a 59 % probability of not playing. This means that we prefer not to play.

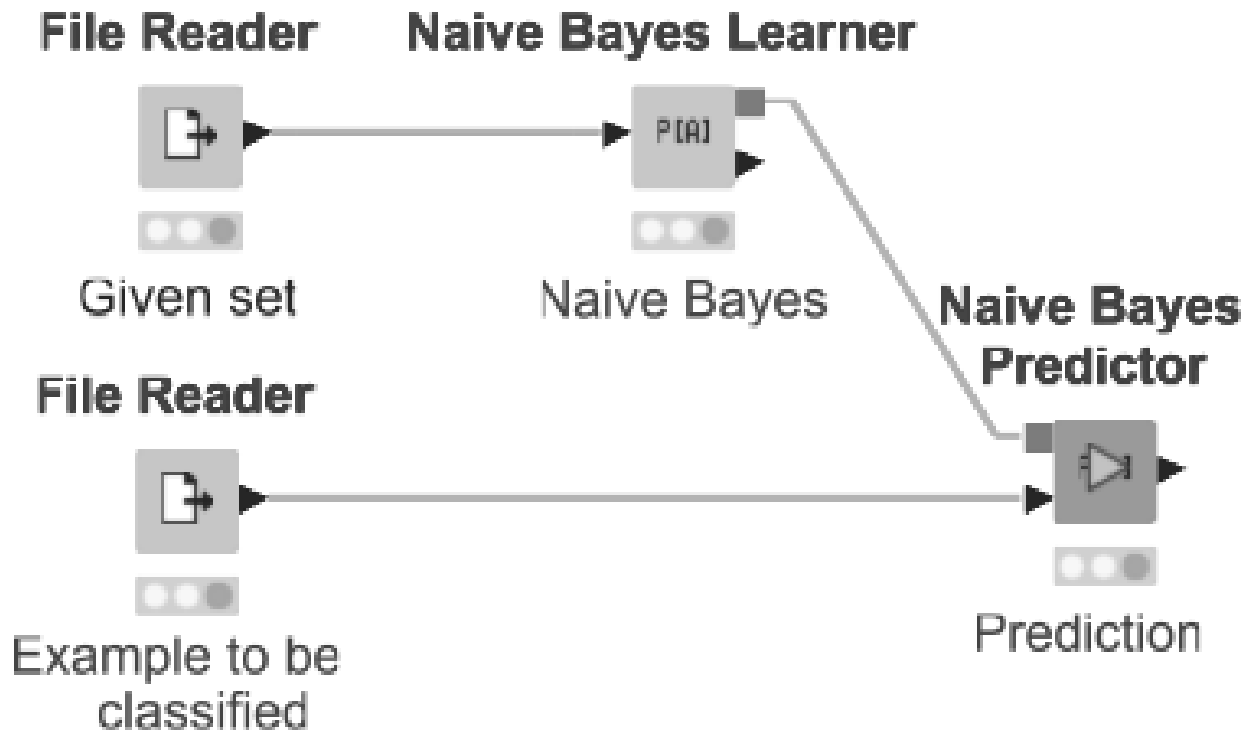


Fig. 5.13 Weather example, workflow with naive Bayes

Classified Test data - 2:16 - Weka Predictor (3.7)							
File							
Table "default" - Rows: 1		Spec - Columns: 7		Properties		Flow Variables	
Row ID	S Outlook	S Temp...	S Humidity	S Windy	D P (Play=no)	D P (Play=yes)	S Predic...
1	sunny	hot	normal	true	0.559	0.441	no

Fig. 5.14 Result obtained with naive Bayes for the weather example

So far, we have assumed that the data is nominal or ordinal. This makes sense, as applying naive Bayes to metric data is nonsensical. Let's look at the weather example (→ Table 5.12) in its original version with the numerical values for *temperature* and *humidity*, see → Appendix A.3.

Tab. 5.12 Weather data, numerical

Outlook	Temp (° F)	Humidity (%)	Windy	Play
sunny	85	85	false	no
sunny	80	90	true	no
overcast	83	78	false	yes
rainy	70	96	false	yes
rainy	68	80	false	yes
rainy	65	70	true	no
overcast	64	65	true	yes
sunny	72	95	false	no
sunny	69	70	false	yes
rainy	75	80	false	yes
sunny	75	70	true	yes
overcast	72	90	true	yes
overcast	81	75	false	yes
rainy	71	80	true	no

If we want to apply naive Bayes to this data, we have two problems:

1. The method considers the temperatures 64°F and 65°F to be different values, even though they are almost identical. This ends up in small numbers for the relative frequencies.
2. The method fails when a prediction is to be made for a temperature of 63°F.

The solution to this problem is to form intervals in advance. This has already been done in → Table 5.10. Although intervals can be formed manually, an automated approach is recommended for large amounts of data. Binning, as explained in → Section 8.2, is suitable for this purpose.

Some variants of the naive Bayes method can handle numerical values without discretizing them. We do not want to go into detail here, but only outline the idea.

We refer to → Table 5.12. How do you determine the relative frequencies for metric values, for example $h(86|\text{no})$? We assume a normal distribution (Gaussian distribution) of the values of the metric attribute and calculate the mean μ and variance σ of the data. For the temperature values *temp*, this is done separately for *play=yes* and *play=no*, of course.

From the respective density function

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

we can now read the corresponding values for h .

The naive Bayes method is characterized by its efficiency. The training set only needs to be run through once to determine the relative frequencies.

One disadvantage is that the method assumes that the attributes are independent, which is usually not the case.

Another disadvantage is that zero values have a kind of veto right. Let's look at the data for movie theater visits shown in → Table 5.13. We have the following attributes: Was it a weekend? Was it a Hollywood movie? How many tickets have already been sold? We also have data on the weather, the genre of the movie, and the length of the movie.

Tab. 5.13 Movie theater visits

Weekend	Hollywood	Tickets	Weather	Genre	Length	Movie theater visit
yes	no	some	good	romance	>120	yes
no	yes	90%	good	action	100-120	no
yes	no	some	good	crime	<90	yes
no	yes	90 %	good	action	90-100	no
no	no	90 %	good	romance	>120	no
yes	no	some	poor	drama	<90	yes
no	no	none	poor	crime	<90	yes
no	yes	some	poor	action	<90	yes
yes	no	90 %	poor	crime	100-120	no
yes	yes	90 %	good	drama	90-100	no
no	no	none	good	action	<90	no
yes	yes	90 %	good	crime	100-120	yes

The relative frequencies are shown in → Table 5.14.

Tab. 5.14 Relative frequencies of movie theater visits

		Visit	
		yes	no
weekend	yes	4/6	2/6
	no	2/6	4/6
hollywood	yes	2/6	3/6
	no	4/6	3/6
tickets	none	1/6	1/6
	some	4/6	0/6
	90 %	1/6	5/6
weather	poor	3/6	1/6
	good	3/6	5/6
genre	romance	1/6	1/6
	action	1/6	3/6
	crime	3/6	1/6
	drama	1/6	1/6
length	<90	4/6	1/6
	90-100	0/6	2/6
	100-120	1/6	2/6
	>120	1/6	1/6

Let us consider the following data record and the corresponding relative frequencies:

	Weekend no	Hollywood yes	Tickets some	Weather good	Genre action	Length 100-120
yes	0.33	0.33	0.67	0.5	0.17	0.17
no	0.67	0.5	0	0.83	0.5	0.33

The zero value for $h[some, no]$ ensures that the likelihood

$L[no](no, yes, some, good, action, 100-120)$

is given the value 0.

This is certainly mathematically correct, but we think it needs to be revised, because the prediction *yes* wins, even though all other relative frequencies for *no* are greater than for *yes*.

We can even exacerbate the situation. To do so, let's look at the following example and the corresponding relative frequencies.

	Weekend no	Hollywood yes	Tickets some	Weather good	Genre action	Length 90-100
yes	0.33	0.33	0.67	0.5	0.17	0
no	0.67	0.5	0	0.83	0.5	0.33

Now no prediction is possible, since in both the *yes* and *no* cases, one of the relative frequencies is 0, so that both likelihoods become 0.

We solve the problem by always adding 1 to the frequencies. This turns the zero values into positive, but still small values. Of course, we have to increase the respective denominator by exactly the number of occurrences. This results in the relative frequencies shown in → Table 5.15.

Tab. 5.15 Relative frequencies of movie theater visits, revised version

		Visit	
		yes	no
weekend	yes	5/8	3/8
	no	3/8	5/8
hollywood	yes	3/8	4/8
	no	5/8	4/8
tickets	none	2/9	2/9
	some	5/9	1/9
	90 %	2/9	6/9
weather	poor	4/8	2/8
	good	4/8	2/8
genre	romance	2/10	2/10
	action	2/10	4/10
	crime	4/10	2/10
	drama	2/10	2/10
length	<90	5/10	2/10
	90-100	1/10	3/10
	100-120	2/10	3/10
	>120	2/10	2/10

We now get a prediction for the combination *no/yes/some/good/action/90-100*: The decision *no* (no movie theater visit) wins with 80 %.

This technique goes back to Laplace. It is implemented in WEKA in the naive Bayes method.

It is not necessary to add a 1. Any positive number x can be added. The denominator must then be increased by $n \times x$, where n is the number of values of the attribute. A weighted variant, i.e., different weights for the values, is also possible.

Task 5.8

(Naive Bayes, restaurant data). → Table 5.16 (see → Task 5.7) contains 12 data records with the following attributes:

- Alternative: Is there another suitable restaurant nearby? {yes, no}
- Fri/Sat: Is it Friday or Saturday? {yes, no}
- Hungry: Am I hungry? {yes, no}
- Guests: How many people are in the restaurant? {none, a few, full}
- Reservation: Have I made a reservation? {yes, no}
- Type: What kind of restaurant is it? {French, Chinese, Italian, Burger}
- Waiting time: How long does the restaurant estimate the wait will be? {0-10, 10-30, 30-60, >60}
- Waiting (target attribute): Am I waiting? {yes, no}

Use naive Bayes to calculate whether we should wait in the last three cases listed in → Table 5.16.

We solve the problem by always adding 1 to the frequencies. This turns the zero values into positive, but still small values. Of course, we have to increase the respective denominator by exactly the number of occurrences. This results in the relative frequencies shown in → Table 5.15.

We now get a prediction for the combination *no/yes/some/good/action/90-100*: The decision *no* (no movie theater visit) wins with 80 %.

This technique goes back to Laplace. It is implemented in WEKA in the naive Bayes method.

It is not necessary to add a 1. Any positive number x can be added. The denominator must then be increased by $n \times x$, where n is the number of values of the attribute. A weighted variant, i.e., different weights for the values, is also possible.

Task 5.8

Task 5.8 (Naive Bayes, restaurant data).

→ Table 5.16 (see → Task 5.7) contains 12 data records with the following attributes:

- Alternative: Is there another suitable restaurant nearby? {yes, no}
- Fri/Sat: Is it Friday or Saturday? {yes, no}
- Hungry: Am I hungry? {yes, no}
- Guests: How many people are in the restaurant? {none, a few, full}
- Reservation: Have I made a reservation? {yes, no}
- Type: What kind of restaurant is it? {French, Chinese, Italian, Burger}
- Waiting time: How long does the restaurant estimate the wait will be? {0-10, 10-30, 30-60, >60}
- Waiting (target attribute): Am I waiting? {yes, no}

Use naive Bayes to calculate whether we should wait in the last three cases listed in → Table 5.16.

Tab. 5.16 Restaurant example data

Alternative	Fri/Sat	Hungry	Guests	Reservation	Type	Waiting time	Waiting
yes	no	yes	a few	yes	French	0-10	yes
yes	no	yes	full	no	Chinese	30-60	no
no	no	no	a few	no	Burger	0-10	yes
yes	yes	yes	full	no	Chinese	10-30	yes
yes	yes	no	full	yes	French	>60	no
no	no	yes	a few	yes	Italian	0-10	yes
no	no	no	none	no	Burger	0-10	no
no	no	yes	a few	yes	Chinese	0-10	yes
no	yes	no	full	no	Burger	>60	no
yes	yes	yes	full	yes	Italian	10-30	no
no	no	no	none	no	Chinese	0-10	no
yes	yes	yes	full	no	Burger	30-60	yes
yes	no	yes	a few	no	French	30-60	
yes	yes	yes	full	yes	Chinese	10-30	
no	no	no	none	no	Burger	0-10	

Task 5.9**Task 5.9 (Naive Bayes, living situation).**

Given a set of data records containing values with the following meanings: e=employee, f=freelancer, m=married, s=single, r=renter, o=owner:

Occupation	e	e	e	e	f	f	f	f	f
Marital status	m	s	m	m	s	s	m	m	m
Children	y	n	n	y	y	n	y	n	n
Debts	y	y	n	y	y	n	y	y	y
Living situation	r	o	r	o	o	r	o	r	r

Where does a single, debt-free employee with children live? Perform the calculation without and with Laplace correction.

5.4 Feedforward neural networks

Classification tasks can also be solved using *artificial neural networks*. Feedforward neural networks, such as *backpropagation networks*, also known as *multilayer perceptrons* (MLP), are used for this purpose.

The development of an artificial neural network requires, on the one hand, the definition of the network structure and, on the other hand, training has a decisive influence on the network behavior. We first consider the network architecture and then discuss possible learning methods.

5.4.1 Architecture

In order to use an artificial neural network for classification, the structure of the network, the network architecture, must first be developed. This concerns the number of neurons in the input

layer, the output layer, the number and size of the hidden layers, and the connections between the neurons (→ Figure 5.15).

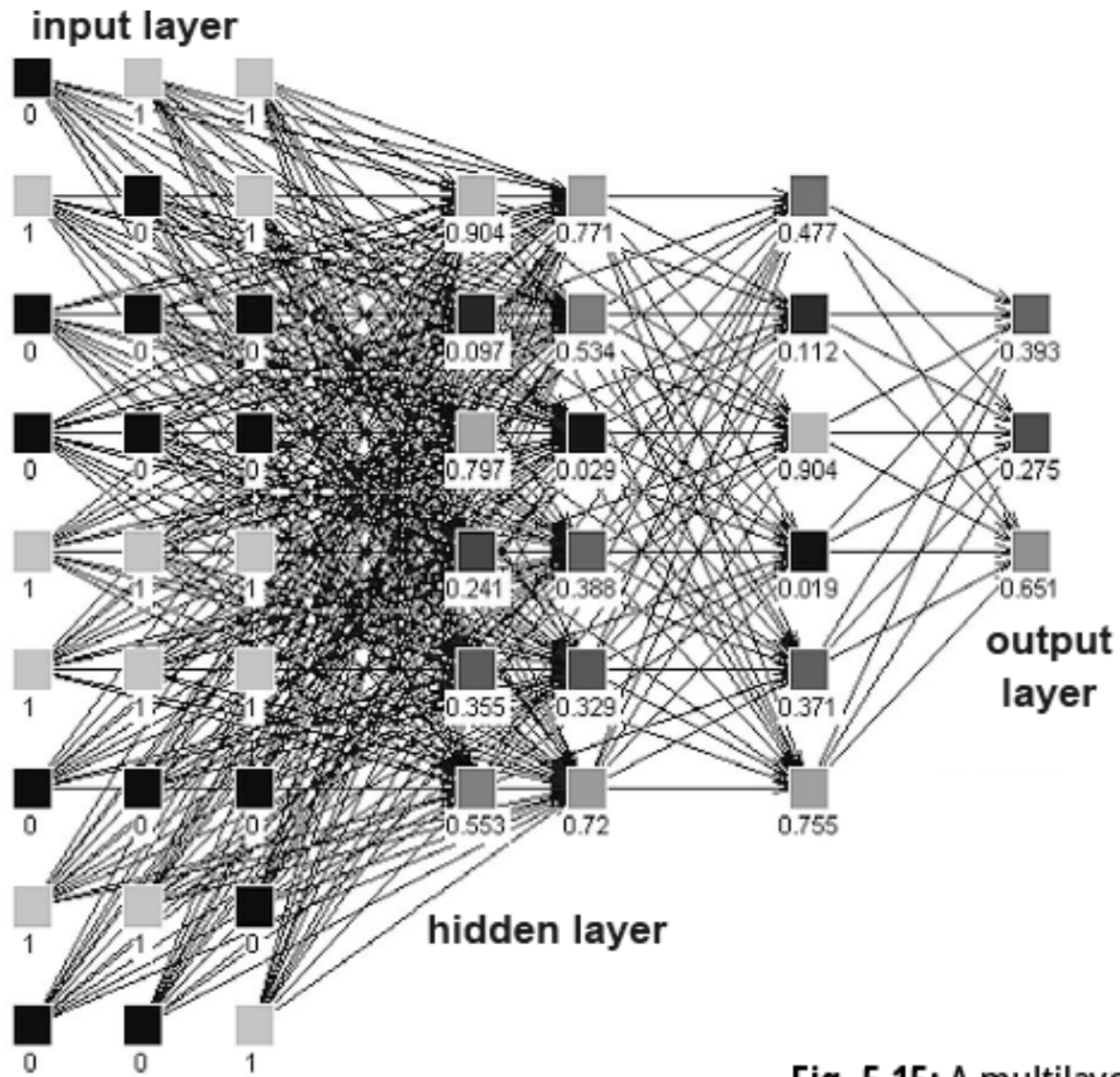


Fig. 5.15: A multilayer

Fig. 5.15 A multilayer feedforward neural network

Let's start with the output layer, see → Figure 5.15 or → 5.16.

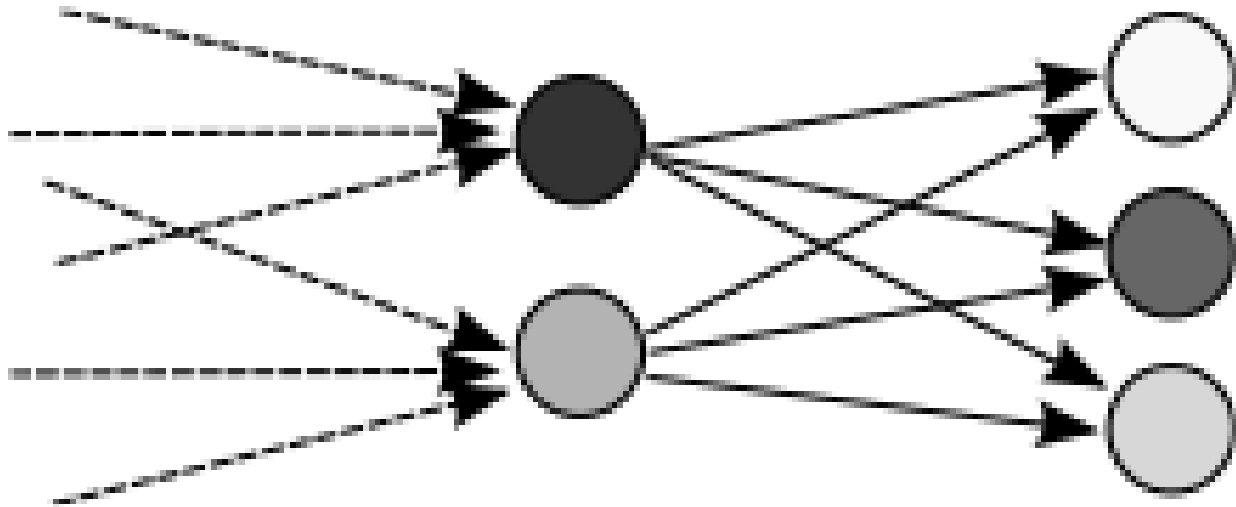


Fig. 5.16 Last trainable connection layer and output layer

Here, the classification result for a data set must be readable. The type of coding of the classification attribute is decisive for the structure of the output layer. In the simplest case, the data is classified into two classes. One neuron may be sufficient for this: *activated* corresponds to one class, *not activated* to the other. An output layer consisting of two neurons is also possible: the activation of a neuron signals membership of the respective class. This approach can be generalized: if a classification into n classes is to be made, an output layer of n neurons is constructed, whereby the activation of a neuron then expresses a measure of membership of the respective class. The class whose neuron is most strongly activated is then predicted.

If the size of the input and output layers is determined by the encoding of the data, the choice of activation function influences the encoding:

1. - If the logistic function is used, the data should be in the range $[0, +1]$. Values outside this range are not meaningful.
2. - The value range $[-1, +1]$ can be used for encoding if the hyperbolic tangent is used as the activation function.

Nominal data is usually processed by a set of neurons. One neuron is used for each possible value: a value is then represented by exactly one 1 and the corresponding number of 0, see also → Section 8.2.5 (p. 214).

Let's consider the type of an item as an example. Possible values are CD, DVD, Blu-Ray, book, and mp3 download. Five different values require five neurons. These values are then coded according to their order:

code(CD)	=(1 0 0 0 0),
code(DVD)	=(0 1 0 0 0),
code(Blu-Ray)	=(0 0 1 0 0),
code(book)	=(0 0 0 1 0),
code(mp3)	=(0 0 0 0 1).

Ordinal values can also be coded in a similar way, using what is known as unarycoding: numbers from 1 to 5 are then coded as sequences of ones. Take the attribute *school grade* as an example. It is coded so that 1 is better than 2, which is better than 3, and so on, i.e., $1 > 2 > 3 > 4 > 5$.

code(1) = (1 1 1 1 1),

code(2)	=(1 1 1 0),
code(3)	=(1 1 0 0),
code(4)	=(1 0 0 0),
code(5)	=(0 0 0 0).

Real numbers require special consideration. It is possible to form intervals. Belonging to an interval is then coded like an ordinal value. Real values can also be normalized and thus mapped to the range $[0, 1]$ (see also → Section 8.2.5). This means that each attribute is processed by exactly one input neuron. It is important to examine whether the differences between the individual values are not blurred too much in the process.

Example 5.7

(Neurons for the body height attribute).

An input for an artificial neural network is to be developed for the body height attribute. The values are in the range 150 to 200. There are many ways to prepare the data so that an artificial neural network can work with it. We present two variants here:

1. - The values are normalized. One neuron is used. The minimum value 150 is then coded as 0.0 and the maximum value 200 as 1.0. The value 180 is then coded as 0.6.
2. - Five intervals are formed and coded using four neurons: Values from 150 to 159 are represented as (0000). The assignment of the neurons with the values (1100) then represents a body height from the interval 170 to 179.

Determining the hidden layer(s) is difficult. While there are clear specifications for the input and output layers based on the data to be processed, this is not the case for the hidden layers. Here, it is necessary to try out several variants: one or two hidden layers with different sizes. Some recommendations based on the authors' experience are as follows: A hidden layer about half the size of the input layer is a good starting point. If the results are not satisfactory, the inner layer is enlarged to the size of the input layer. If the result is still not as desired, a second inner layer increases the performance of the network. Theoretically, more than two inner layers are not necessary (see [→ Zel97]), but in practice, neural networks with many inner layers, which are used in deep learning, have led to new successful applications.

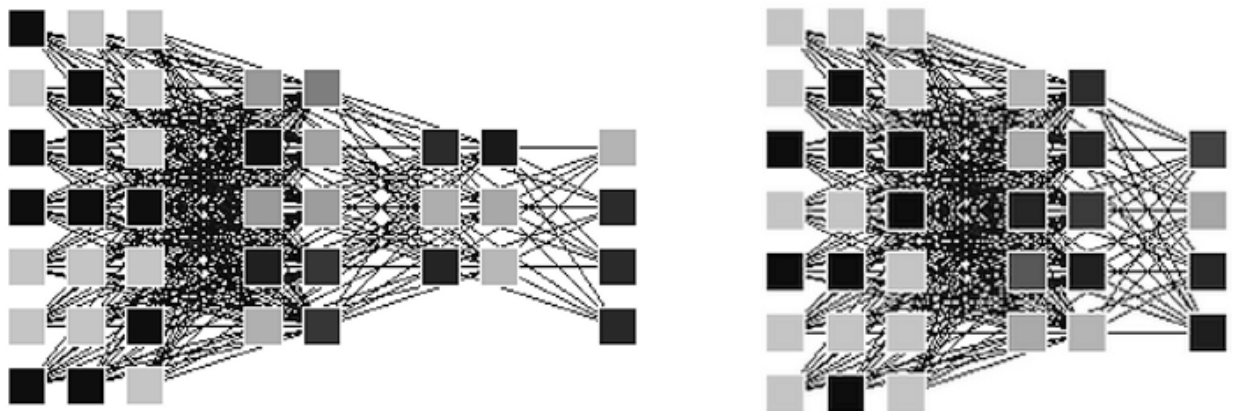


Fig. 5.17 Different hidden layers

5.4.2 The backpropagation of error learning method

Once the architecture of a neural network has been defined, the training data must be encoded: The number of values per data record must correspond to the number of neurons in the input layer. The encoding of the classification attribute must be representable with the output layer. To train a classifier, in addition to the input data, the desired results for the training data are also required. This is *supervised learning* of an artificial neural network. If the result is known, the error in the behavior of a neural network can be determined. This error is then used to modify the artificial neural network. In this way, the neural network learns the desired classification behavior.

Feedforward neural networks are trained using a *backpropagation-of-error algorithm*. A feedforward neural network consists of an input layer, one or more layers of hidden neurons, and an output layer. → Figure 5.18 shows the structure. The neurons in one layer are usually connected to all neurons in the subsequent layer. Connections only exist from the input layer to the first hidden layer, from there possibly to a subsequent intermediate layer, and then from the last intermediate layer to the output layer. This is where the name comes from: *feedforward neural network*. The individual layers consist of several (many) artificial neurons.

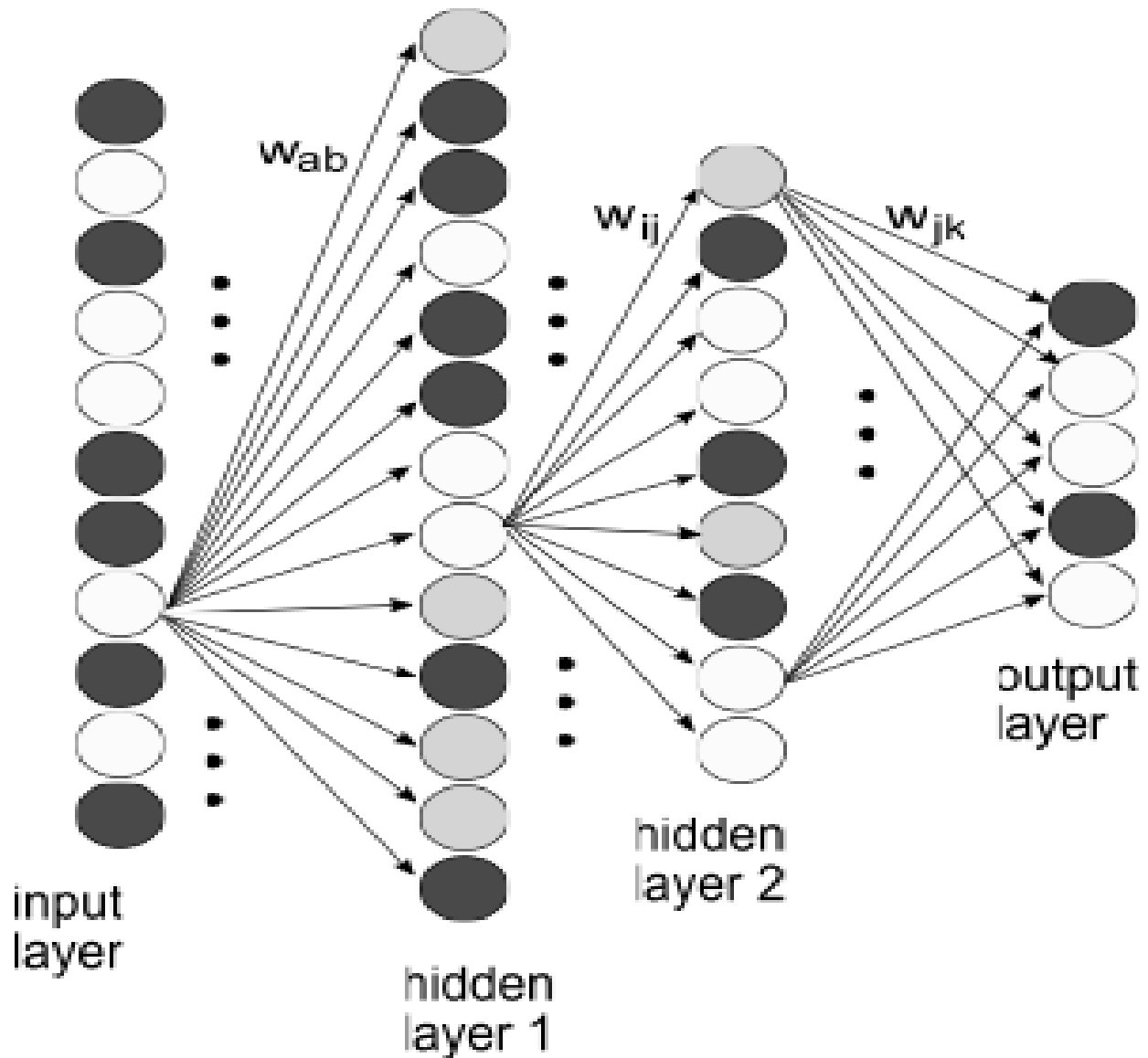


Fig. 5.18 Layers in a feedforward neural network

As is usual with classification tasks, the network is trained using a set of training data. The *power* of a network, the stored knowledge, is hidden in the weights of the connections and in the threshold values of the neurons. Training, or learning, therefore consists of changing the weights and threshold values. As a result, the *error* of the neural network is reduced. The *network error* is calculated from the deviation between the output values of the output neurons O_k and the expected training values t_k :

$$err = \sum_k (o_k - t_k)^2 \quad 5.16$$

We can now imagine that the weights of the connections to the output neurons are changed according to the calculated network error. Let's consider simplified binary values in the following.

Suppose a value of 1 is expected, but the corresponding output neuron is not active (output is 0). In this case, the connection weights to this neuron must be increased to make the network

input of the output neuron higher, thereby increasing the chance that the neuron will be activated. Conversely, the weight value must be reduced if the output neuron was activated but a value of 0 is expected.

The exact derivation and mathematical justification can be found in [→Zel97] or [→LC23]. The change in a weight w_{ij} is calculated according to

$$\Delta w_{ij} = \lambda \cdot o_i \cdot \delta_j \quad 5.17$$

where the terms have the following meanings:

- λ is the *learning factor*, which determines how strongly the error changes the weights. The choice of learning factor can have a decisive influence on learning success. Typically, a learning factor in the range 0.1 to 0.8 is used.
- o_i is the output value of the predecessor neuron of the connection to neuron j .
- δ_j is the error signal, which is determined differently for the neurons of the hidden layers than for the neurons of the output layer. Since the output and thus the error of a neuron also depends on the selected activation function, the activation function, or more precisely its first derivative, is included in the calculation of the error signal.

The change in weights depends on the selected activation function. If the logistic function is used, the error signal for an output neuron j is

$$\delta_j = o_j \cdot (1 - o_j) \cdot (t_j - o_j) \quad 5.17$$

The error signal is not only determined by the difference between the calculated output O_j and the desired output, the training value t_j . The activation function has a significant influence on the output of a neuron and thus also on the resulting error. Put simply, the increase in the error curve is now taken into account when calculating the error signal. The increase of a curve at a point is calculated using the first derivative, and so the first derivative of the logistic function is included in the calculation.

Task 5.10

Task 5.10 (Logistic activation function).

Make sure that the first derivative of the logistic activation function has the form

$$f'_{act}(x) = f_{act}(x) \cdot (1 - f_{act}(x))$$

The error signal of a neuron in a hidden layer is not so easy to calculate, since there is no expected value or training value for a hidden neuron. This means that no error can be determined from the difference between a calculated value and an expected value.

How can the weights in the connection layers to the hidden neuron layers and the threshold values of these hidden neurons still be adjusted? Consider how the error arises in the output layer. Of course, the error arises from *incorrect* activations of the output neurons. However, this value is calculated from the network input of a neuron, which in turn is derived from the outputs of the preceding neurons multiplied by the respective connection weights. According to this consideration, the error signal of a neuron j in a hidden layer is determined taking into account all connections w_{jk} to subsequent neurons k and their error signals δ_k .

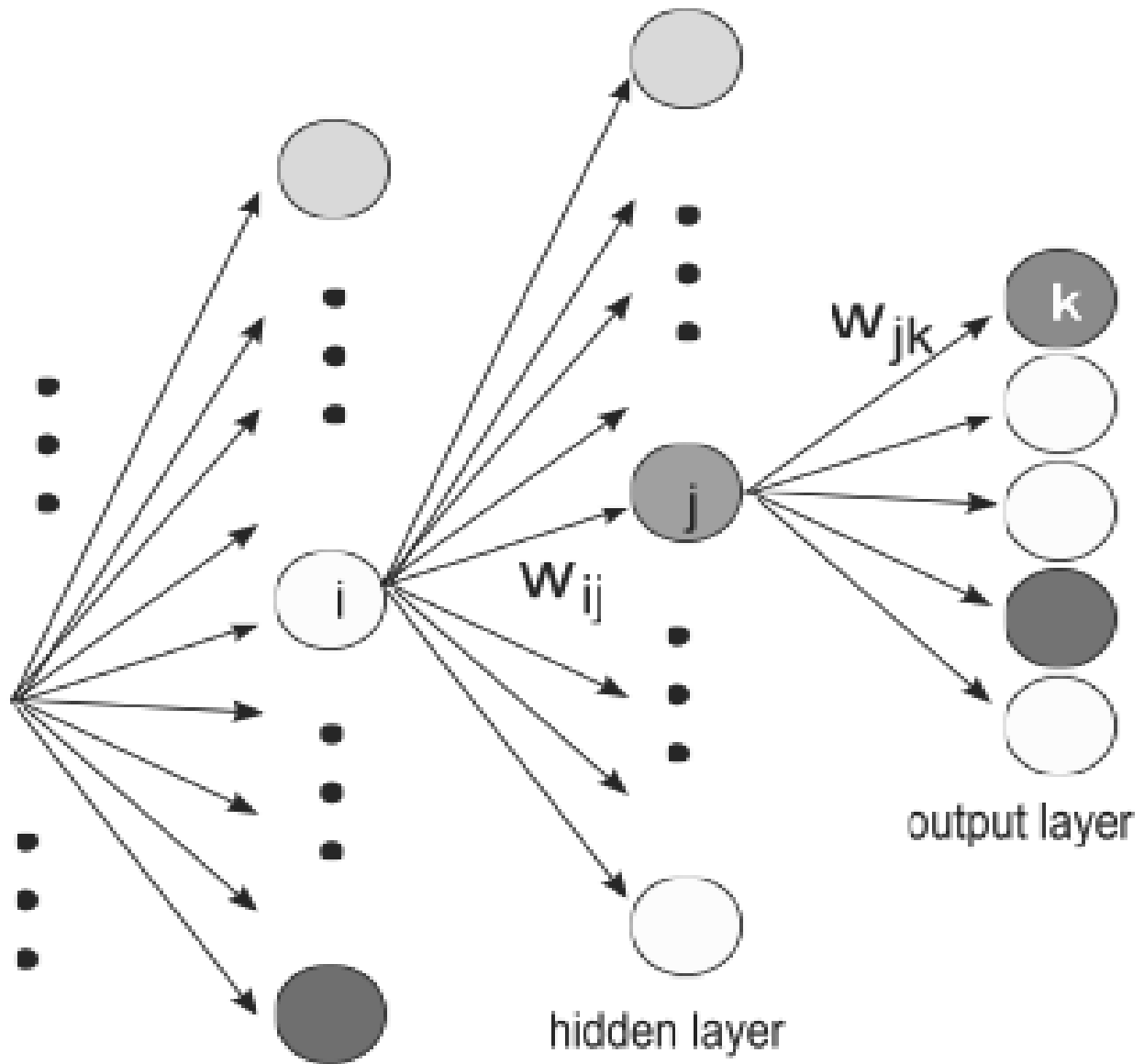


Fig. 5.19 Influence of subsequent neurons on the error signal

$$\delta_j = \begin{cases} o_j \cdot (1 - o_j) \cdot (t_j - o_j) & \text{if } j \text{ output neuron} \\ o_j \cdot (1 - o_j) \cdot \sum_k \delta_k \cdot w_{jk} & \text{if } j \text{ hidden neuron} \end{cases} \quad 5.18$$

The influence of the activation function on the error signal is the same for both types of neurons. The error signal of a neuron is now used to change the weight of a connection to neuron j :

$$w'_{ij} = w_{ij} + \lambda \cdot o_i \cdot \delta_j \quad 5.19$$

First, only the error signal for the neurons of the output layer can be calculated. Once the error signals of the neurons of the output layer are known, the error signals of the last hidden layer can be calculated, and so on until the first hidden layer. This is why this learning algorithm is called *back propagation of error*.


```

PROCEDURE Backpropagation of Error
    nCycles := 0
    REPEAT
        error := 0
        nCycles := nCycles + 1
        Input pattern i;
    // Calculate the error signal  $fsig_j$  for each output neuron  $j$ :
    FOR j := 1 ... numberOfNeurons(OutputLayer) DO
        Calculate output  $o_j$ 
        Calculate error  $f_j := t_j - o_j$ 

        calculate error signal  $fsig_j := o_j \cdot (1 - o_j) \cdot f_j$ 
        error := error +  $f_j^2$ 
    END FOR
    // Go through layer by layer from back to front:
    FOR s := OutputLayer-1 DOWNTO FirstLayer DO
    // Calculate error signal  $fsig_{sm}$  of the neuron  $k$  in layer  $s$ :
    FOR k := 1 ... numberOfNeurons(Layers) DO
        // Calculate weighted sum of
        // error signals of the subsequent neurons of Layer s
        + 1

         $fsum := 0$ 
        FOR m := 1 ... numberOfNeurons(Layer(s+1)) DO
             $fsum := fsum + w_{km} \cdot fsig_{(s+1)m}$ 
        END FOR
        // calculate error signal  $Neuron_{sk}$  from the derivative
        of the activation function
        // and the weighted sum of the error signals of the
        successors:
         $fsig_{sk} := o_{sk} \cdot (1 - o_{sk}) \cdot fsum$ 
        // Adjustment of weights to the subsequent neurons:
        FOR m := 1 ... numberOfNeurons(Layer(s+1)) DO
             $w_{km} := w_{km} + learningRate \cdot o_{sk} \cdot fsig_{(s+1)m}$ 
        END FOR
    END FOR
    END FOR
UNTIL error tolerableError OR number of cycles reached
END Backpropagation of Error

```

Listing 5.9 (Backpropagation-of-error algorithm).

5.4.3 Modifications to the backpropagation algorithm

If we consider the architecture of a neural network to be largely fixed, the result, i.e., the quality of the network, depends on the training process. The training process is influenced by the method used and the parameter values applied. With the simple back-propagation algorithm, different

values for the learning parameter λ can be tried. The goal is to reduce the network error or to reduce the number of learning steps required to achieve this minimum error.

Since the backpropagation algorithm is a gradient descent method, situations may arise that delay or even prevent network adaptation. If we simplify the network error as a function of a single connection weight, the problems can be represented graphically, see → Figure 5.20.

However, the error curve is not known, so the size of the adjustment steps can only be determined experimentally using the learning parameter. If the learning parameter is small (close to 0) and there is a plateau in the error curve, the learning process is significantly delayed. This can even lead to learning being terminated, as a reduction in the network error is no longer expected.

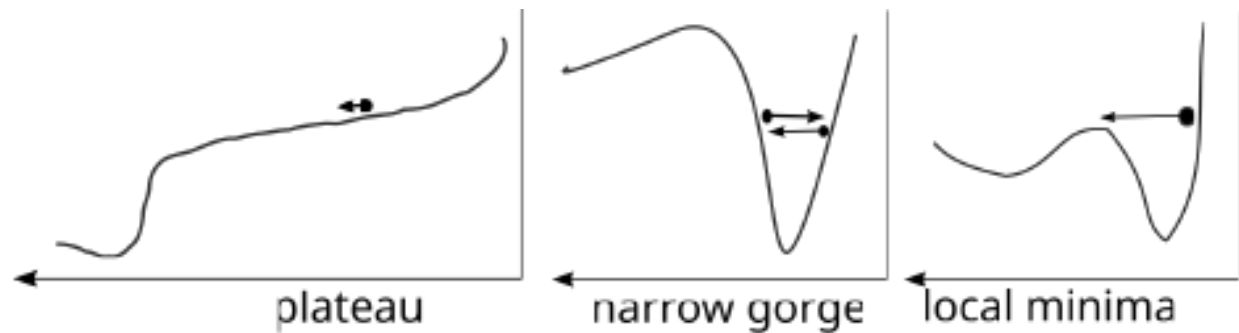


Fig. 5.20 Difficult situations in the learning process

A large learning parameter (close to 1) causes the error curve to oscillate in narrow gorges: figuratively speaking, the error value jumps from one wall to the other and back again: it does not settle at the minimum.

Similarly, a large step size can result in the global minimum being skipped and only a local minimum being reached, as shown in the the right-hand part of → Figure 5.20. These problems have led to the development of several modifications to the backpropagation algorithm, of which we present three commonly used ones below.

Backpropagation with momentum

The new connection weight between two neurons at time $t + 1$ is derived from the current weight at time t , taking into account the change resulting from backpropagation. In addition, the previous change at time $t - 1$ is included, with the parameter μ controlling the extent to which this occurs.

$$\Delta w_{ij}(t) = \lambda \cdot o_i \cdot \delta_j \quad 5.20$$

$$w_{ij}(t + 1) = w_{ij}(t) + \Delta w_{ij}(t) + \mu \cdot \Delta w_{ij}(t - 1) \quad 5.21$$

QuickProp

Graphically illustrated, the backpropagation method as a gradient descent method determines the new weight by drawing the tangent line, to the point of the error curve. QuickProp attempts to reach the minimum error more quickly by determining a quadratic curve. The last two weight values, $w^{ij}(t - 2)$ and $w^{ij}(t - 1)$, are used to determine the parabola. QuickProp often generates an oscillating error curve during the learning process.

Resilient Propagation (RPROP)

Resilient Propagation completely dispenses with directly determining the weight change from the calculated error, instead focusing on the direction of the error. The amount of the change and the direction of the change are calculated separately:

- If the last two changes have the same sign, then a large adjustment step is made with this sign: factor $1 < \eta^+$, often $\eta^+ = 1.2$.
- If different directions are calculated for the penultimate and last changes, the last change is reversed and a smaller adjustment step is made: factor $\eta^- < 1$, often $\eta = 0.5$.

Resilient propagation is used in the *MLP* node of KNIME.

In practical applications, the simple backpropagation method, also known as *vanilla backpropagation*, often provides the best results. Readers can find more information on the mathematical foundations of the backpropagation algorithm and the various modifications in [→ LC23] and, in particular, in [→ Zel97].

5.4.4 An example

When introducing the JAVANNNS software in → Section 1.6.3, one example was already described. While the RPROP learning method is preset in KNIME, all learning methods described in the previous Section can be used and tried out with JAVANNNS. Here, we focus on working with the neural network in an example, rather than on the data mining problem in general.

Tab. 5.17 A data set for classification

0	0	1	0	0	1	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	1	1	1
0	1	1	0	1	0	0	1	0	0	0	1	0	1	1	0	1	0	0	0	1	0	0	0	1	1	1	1	2
0	1	1	0	1	0	0	1	0	0	0	1	0	0	1	0	0	0	0	1	1	0	0	1	0	1	1	0	3
1	0	0	0	1	0	0	0	1	0	0	0	1	0	1	0	1	1	1	1	0	0	1	0	0	0	1	0	4
1	1	1	1	1	0	0	0	1	0	0	0	1	1	1	0	0	0	0	1	0	0	0	1	1	1	1	1	5
0	1	1	0	1	0	0	1	1	0	0	0	1	1	1	0	1	0	0	1	1	0	0	1	1	1	1	1	6
1	1	1	1	0	0	0	1	0	0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0	7
0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0	8
0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	1	0	0	0	1	1	0	0	1	0	1	1	0	9
0	1	1	0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	0	1	1	0	0

It is difficult to discern the type of classification from → Table 5.17. There are 28 binary input values and an integer classification feature. However, if a row is interpreted as a 4x7 matrix, the inputs can be easily recognized as pixel representations of the 10 digits (→ Figure 5.21).

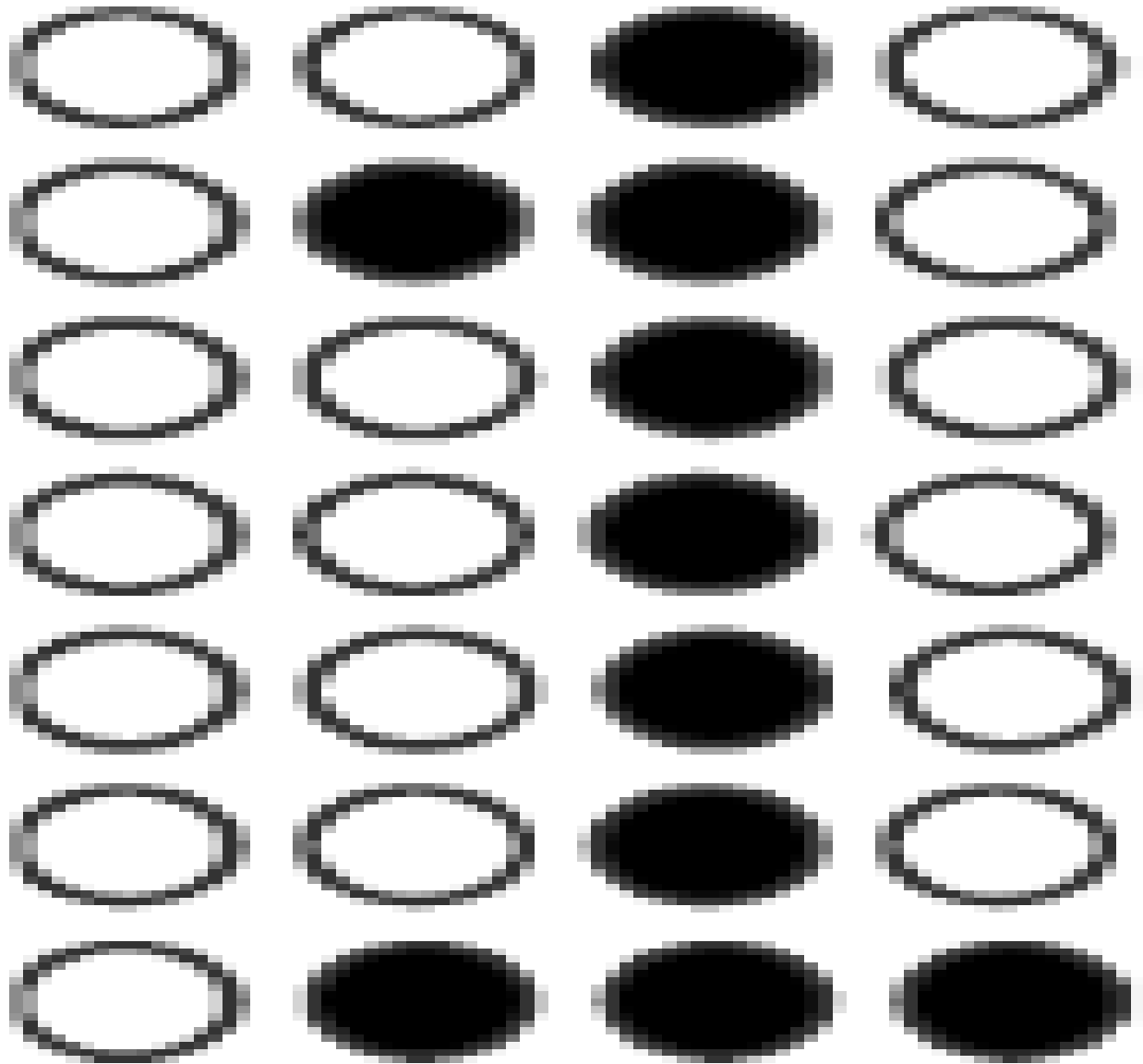


Fig. 5.21 Input pattern: digit 1 in pixel representation

We use the neural network for character recognition. Character recognition is also pattern recognition, as in data mining, except that we as humans know the pattern here and can therefore quickly see the result of the classification.

Here, too, the goal is not only to recognize the training data, but also to classify new, unknown data. To do this, we have slightly *destroyed* the digit patterns by adding noise to them. The network must then be able to recognize these noisy patterns as well only then will the classification task be successfully completed.

A network of 4x7 input neurons is created, and the 10 classes of the classification feature result in 10 output neurons. It is expected that exactly one neuron is always activated, thus indicating the class associated with the pattern. We try using five hidden neurons as the intermediate layer. The pattern file is created as described in → Section 1.6.3, and a copy of this pattern file is generated with correspondingly noisy inputs.

Once the network has been created and the pattern files loaded, training of the network can begin. In JAVANNs, the network error for the patterns in the training file can be easily compared with the network error for the patterns in the test file. To do this, the respective file must be defined as the *training set* or *validation set* in the *Pattern* tab of the *control panel*. The result can be tracked in the *error graph* window during training.

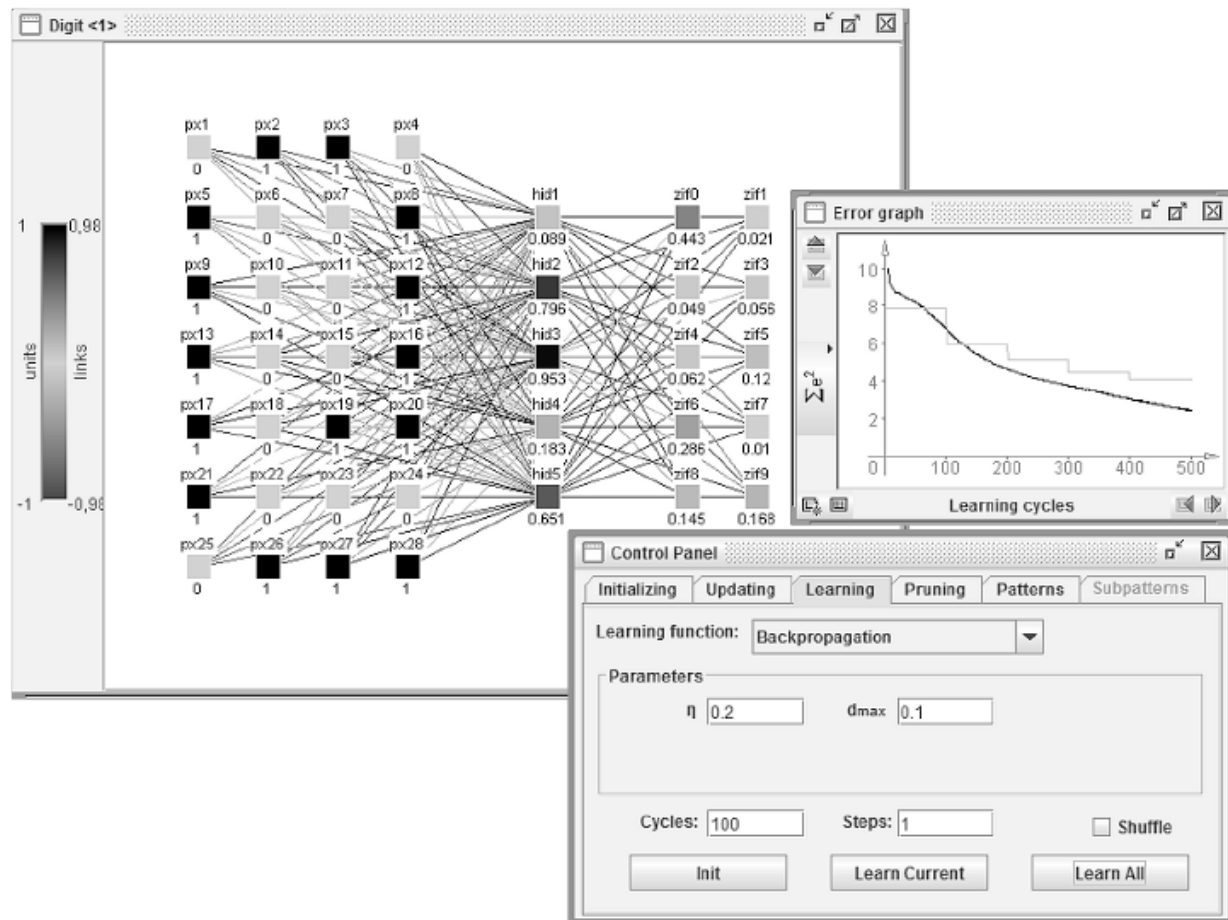


Fig. 5.22 Training the neural network in JAVANNs

→ Figure 5.22 shows both the network architecture and the training process. Training was performed using the standard parameters for the backpropagation algorithm. The error curve shows that training using the training set also leads to a reduction in the error in the test set. Further training reduces the error for the training set, but increases the error for the test set. This is a clear sign of excessive learning, known as *overfitting*. In our example, not all noisy patterns in the test set are recognized correctly. This is not surprising given that 10 % of the pixels in a 4x7 matrix are faulty. Is the pattern in → Figure 5.23 a slightly noisy 6 or an 8?

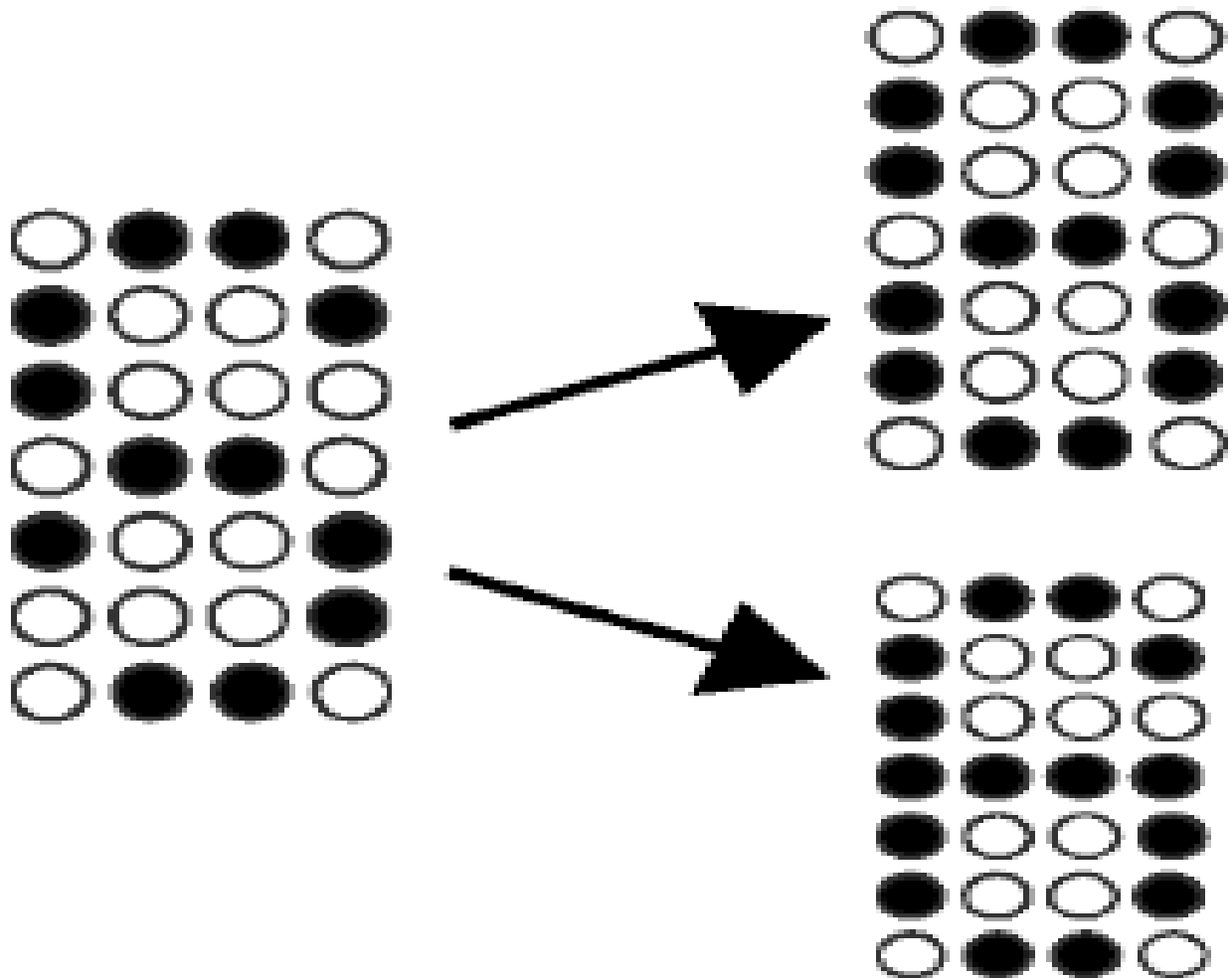


Fig. 5.23 Mapping a noisy pattern

This incorrect classification of the pattern is a typical feature of data mining: despite good preparatory work, our model will rarely be able to classify all future data sets clearly and correctly. In this small example, the feedforward neural network developed is not only able to recognize error-free digit representations, but can also correctly classify a whole series of slightly noisy pixel representations into the appropriate class.

Examples of point classification in the plane can be performed using the interactive tool from TensorFlow (→ <https://playground.tensorflow.org>). Here, the performance of a neural network is clearly illustrated depending on the number and size of the hidden layers. The two input neurons X_1 and X_2 are assigned the x, y coordinates of a point in the plane. Various geometric distributions of the two classes can now be selected and the recognition rate can be influenced by varying the hidden layers, see → Figure 5.24.

Task 5.11

Task 5.11 (Character recognition).

Develop a feedforward neural network that is capable of recognizing digits in pixel representation. You can use the JAVANNNS example for recognizing capital letters as a model.

Task 5.12

Task 5.12 (Iris classification using feedforward neural networks).

Develop an artificial neural network that can classify Iris data. Create the network in KNIME, JAVANNNS, Tensor Flow Playground, or implement the network in Python. What are the advantages and disadvantages of each system? The task and data are described in → Appendix A.1.

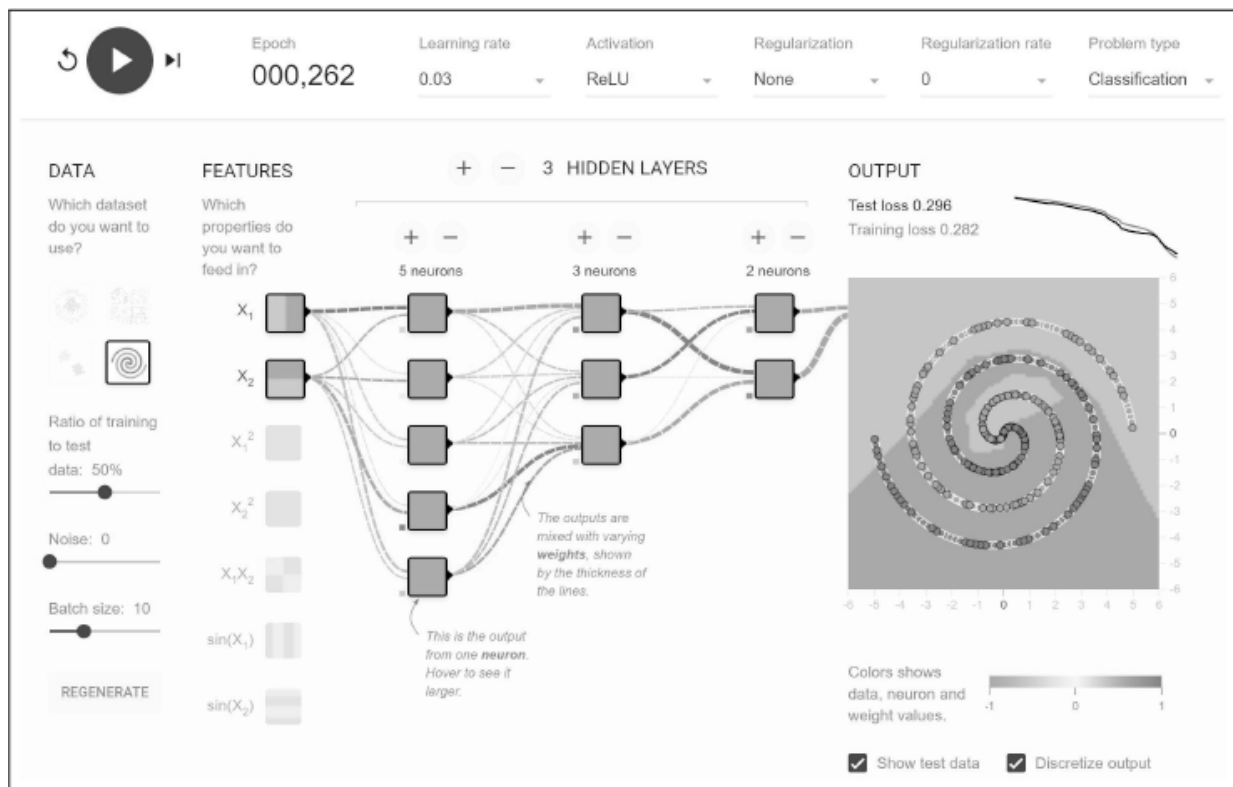


Fig. 5.24 Recognition of spiral-shaped classes using TensorFlow

Task 5.13

Task 5.13 (Artificial neural networks for restaurant visits).

Develop a feedforward neural network to decide whether to wait for a free table in a restaurant. Use the data in → Table 5.16, p. 117.

- What preprocessing of the data is necessary?
- Define the architecture of the network. Try different sizes for the inner layers.
- Use KNIME and JAVANNNS to train the network.
- How does the trained network respond to these situations?

Alternative	Fri/Sat	Hungry	Guests	Reservation	Type	Waiting time	Waiting
yes	no	yes	a few	no	French	30-60	
yes	yes	yes	full	yes	Chinese	10-30	
no	no	no	none	no	Burger	0-10	

5.4.5 Convolutional neural networks

Convolutional neural networks (CNN) are a special form of feed forward neural networks that are used in particular for recognizing objects in images. When an image is mapped pixel by pixel with color information onto neurons, very large layers are created. If these are fully interconnected, this results in millions of connections: Let's assume a small image measuring 30x30 pixels, with color information stored in three values for each pixel, resulting in a layer size of $30 \times 30 \times 3 = 2700$ neurons. If we now assume only one hidden layer of 400 neurons and one classification layer of 10 neurons, we end up with more than a million connection weights.

On the one hand, this leads to long computing times during training and, more importantly, to a network with very high storage capacity: the network learns by heart and becomes overfitted. Although such a network learns the training data very well, it is unable to process new, unknown patterns correctly.

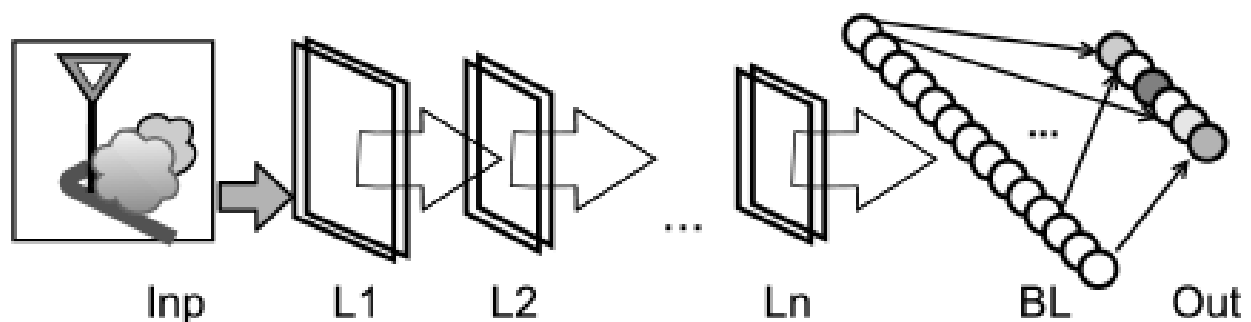


Fig. 5.25 Architecture of a convolutional neural network

In a *convolutional neural network* CNN, so-called convolutions are performed. In this process, a small matrix, known as a filter, is moved step by step over the image and the result is stored in a new layer. This is followed by a reduction in dimensions using a *pooling filter*, usually max pooling, but sometimes average pooling. These two steps can be combined into one layer (see layers L_i in → Figure 5.25). If several or even many such layers are connected in series, this is referred to as *deep learning*. A fully connected backpropagation network (see layer BL in → Figure 5.25) is added as the final processing layer, which then displays the classification in the output. A softmax algorithm is used to identify the class. The class assigned to the neuron with the highest activation is predicted.

A more detailed description of the architecture and the learning process for using CNN in image recognition can be found in the literature, for example [→ SS25], [→ LC23] or [→ CW25].

5.5 Support Vector Machines

5.5.1 Basic principle

Support vector machines (SVM) are powerful classification tools. The idea can be illustrated well using a two-dimensional example. Let us consider the set of points shown in → Figure 5.26.

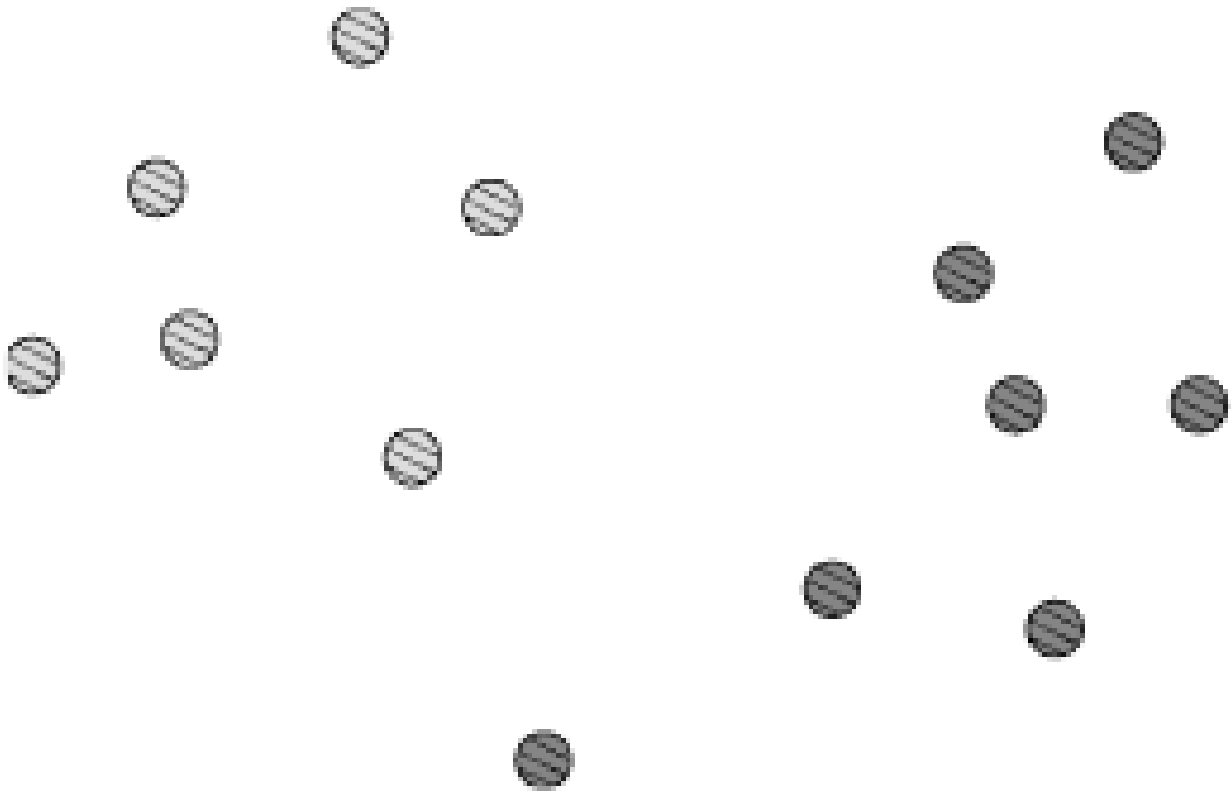


Fig. 5.26 Two-dimensional set of points

The points belong to two classes, which are indicated in the image by different gray values.

How can a classifier be generated from such an example? The SVM separates the given points with a straight line, as shown in → Figure 5.27.

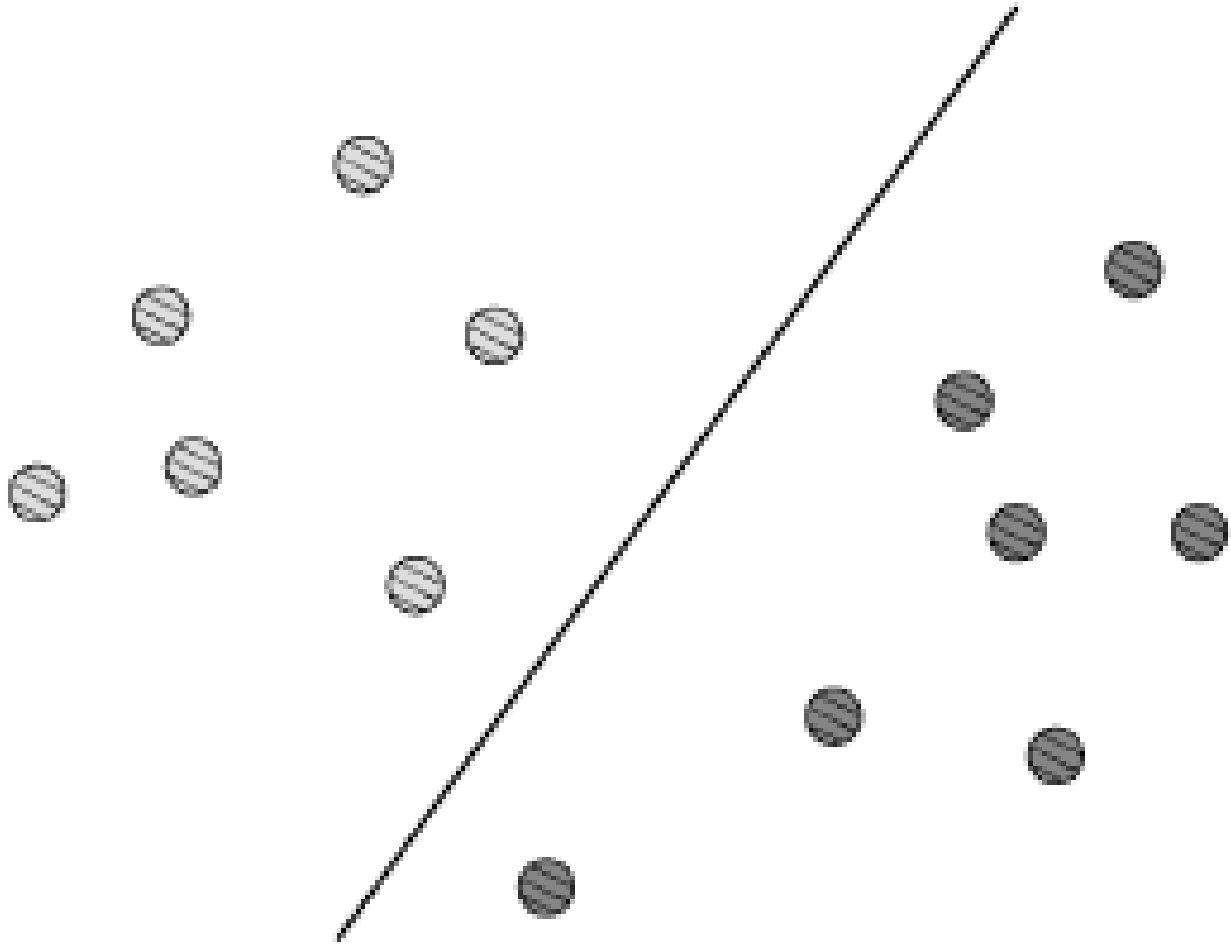


Fig. 5.27 Separating classes using SVM

When new data is to be classified, the SVM checks which side of the line the data lies on.

What types of data can an SVM handle? Obviously, only metric data can be processed.

However, we generally don't just have examples with two attributes, i.e., points in the plane, but rather multiple attributes, often significantly more than two. How can the concept of SVM be generalized to n -dimensional space $\mathbb{R}^n$? To answer this question, let us first consider how an SVM works in three-dimensional space. Given points in $\mathbb{R}^3$, we want to separate them again according to their class membership. This can no longer be done with a straight line; a plane is required, see → Figure 5.28.

This principle can be generalized to multidimensional space $\mathbb{R}^n$ with n attributes. In n -dimensional space, the given set of points is divided by a so-called *hyperplane*, an $n-1$ -dimensional plane. For illustrative purposes, however, we will focus on examples in two- or three-dimensional space.

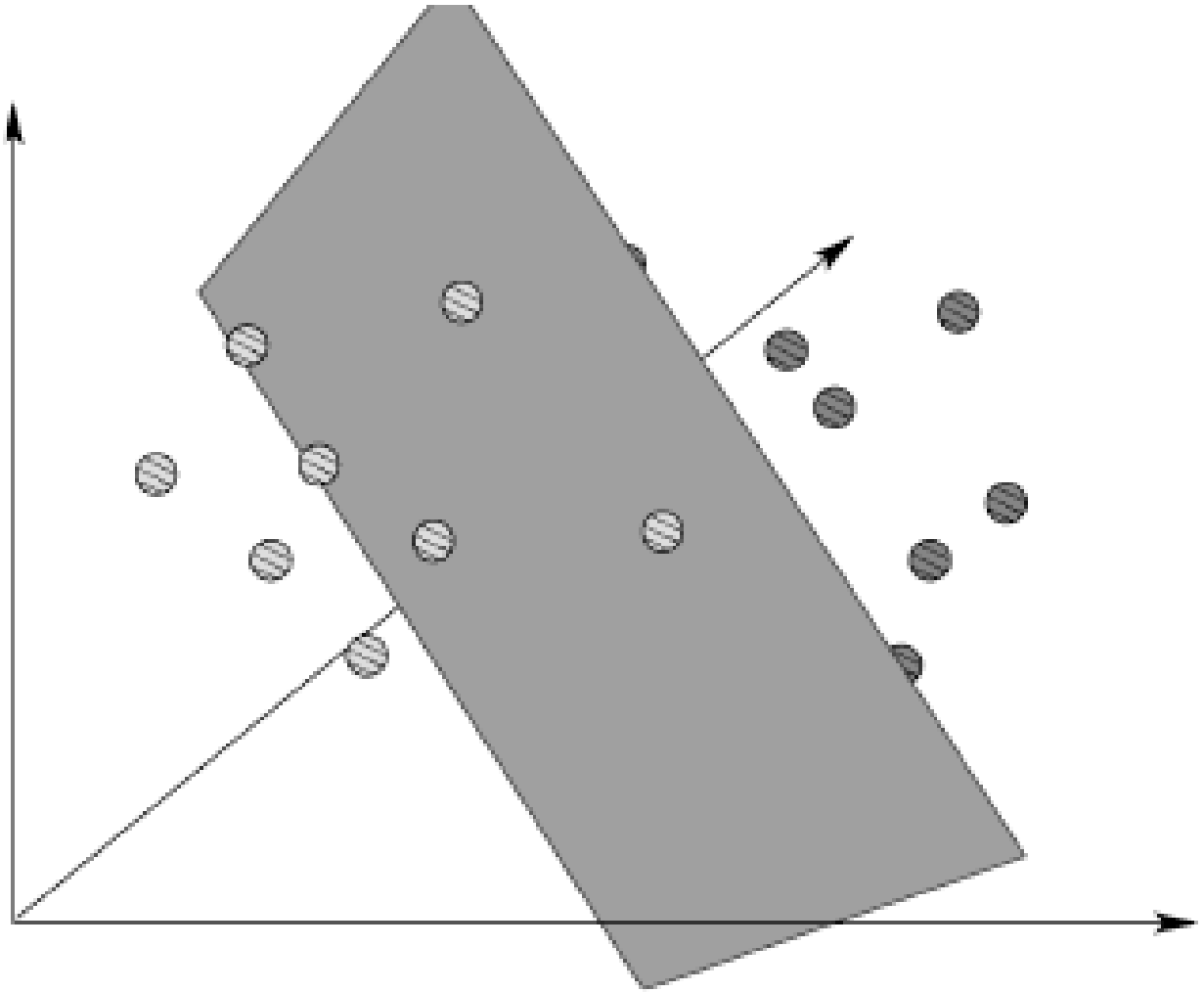


Fig. 5.28 SVM in three-dimensional space

An SVM in the variant presented here handles classification tasks with exactly two classes.

5.5.2 Formal representation of support vector machines

Given is a training set

$$\{(x_1, k_1), (x_2, k_2), \dots, (x_m, k_m)\}$$

where $x_i \in \mathbb{R}^n$ and $k_i \in \{1, -1\}$.

The given x_i are n -tuples that are assigned to two different classes, which we encode here as -1 and 1.

The goal of SVM is to find a hyperplane that *best* separates the two classes. What does *best* mean in this context? We are looking for the hyperplane whose minimum distance to the elements of the two classes is maximized.

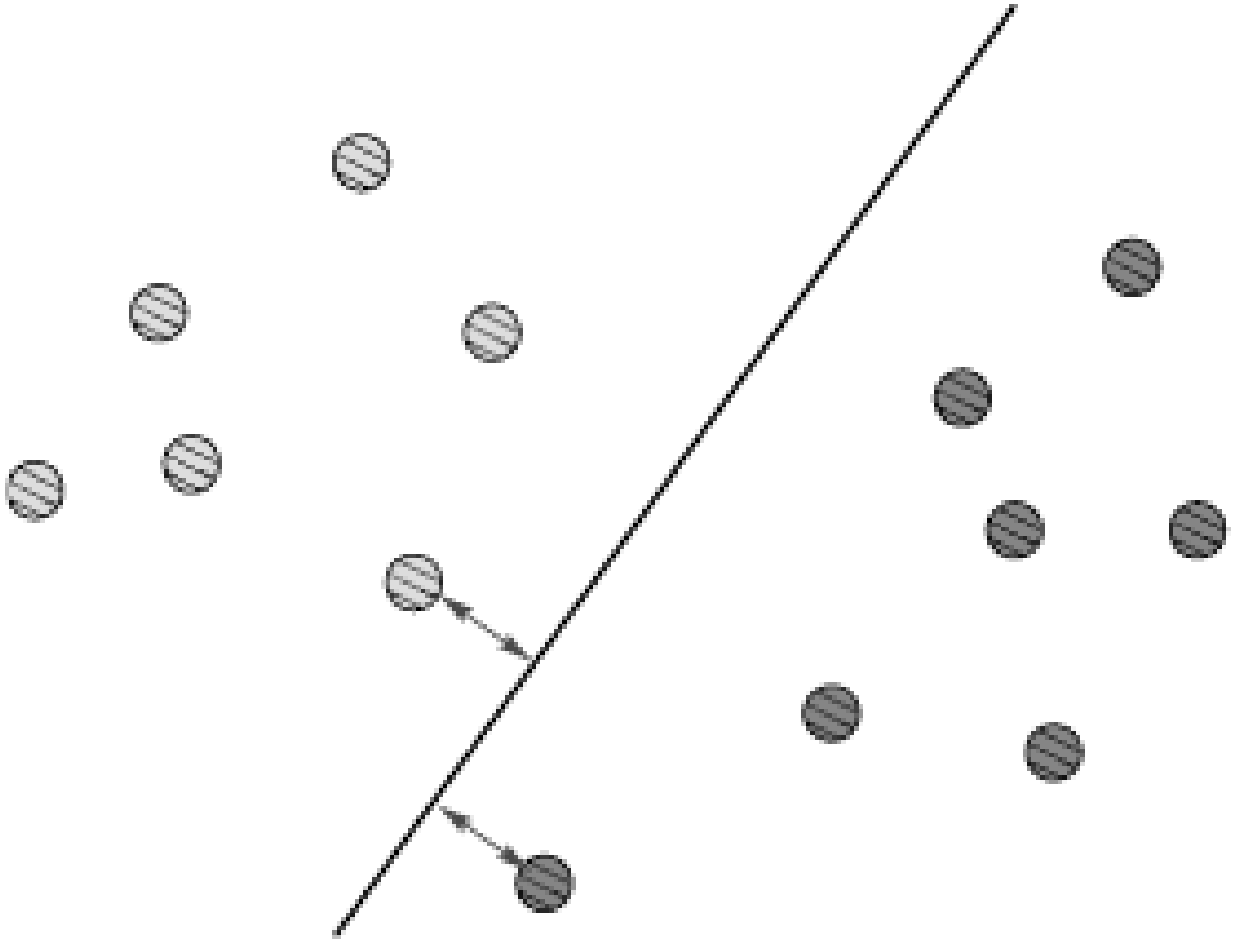


Fig. 5.29 SVM, optimal hyperplane

Such a hyperplane can be described by the equation

$$w^T \cdot x + b = 0 \quad 5.22$$

where w is a vector perpendicular to the hyperplane. $w^T \cdot x$ is the scalar product of two vectors, i.e.

$$\sum_{i=1}^n w_i \cdot x_i \quad 5.23$$

The above representation $w^T \cdot x + b = 0$ is equivalent to the so-called point normal form:

$$w^T \cdot (x - x_0) = 0 \quad 5.24$$

x_0 is a point on the hyperplane. All other points x on the hyperplane satisfy the condition that the scalar product formed by the vector $(x - x_0)$ and vector w is zero, since these are obviously perpendicular to each other. The vector w is also referred to as a *support vector*, from which the name of the method is derived. Once such an optimal hyperplane has been found, the class of a new point x is simply determined by the sign of the term $w^T \cdot x + b$. If the sign is positive,

$$w^T \cdot x + b > 0$$

x is assigned to class 1, otherwise it is assigned to class -1. The sign of $w^T \cdot x + b$ therefore determines the class assignment:

$$\text{class}(x) = \begin{cases} 1 & \text{if } w^T \cdot x + b > 0 \\ -1 & \text{else } w^T \cdot x + b < 0 \end{cases}$$

Or, more concisely:

$$\text{class}(x) = \text{sgn}(w^T \cdot x + b) \quad 5.26$$

Obviously, we cannot make any predictions for points lying on the hyperplane.

Note 5.10

(Linear separability). So far, we have implicitly assumed that there exists a hyperplane that accurately separates all given examples. If this is the case, we speak of linear separability.

How do we determine an optimal support vector w ? In the case of linear separability, we choose the support vector such that the quadratic norm $\|w\|$ is minimized under the restriction that

$$k_i \cdot (w^T \cdot x_i + b) \geq 1 \quad 5.27$$

applies for all $i = 1 \dots m$.

In most cases, linear separability does not exist. There can be various reasons for this. On the one hand, our data may be noisy, meaning that it is not accurate. On the other hand, the classes may overlap. In this case, it must be allowed that the restrictions are violated. However, the violation of the restrictions should of course be as small as possible. Therefore, a positive slack variable ζ_i is introduced for each of the m restrictions:

$$k_i \cdot (w^T \cdot x_i + b) \geq 1 - \zeta_i \text{ for all } 1 \leq i \leq m \quad 5.28$$

At this point, we will not go into detail about how this optimization problem is solved. The SVM converts the optimization problem into a dual problem using Lagrange multipliers and Karush-Kuhn-Tucker conditions, and then solves it.

There are extensions of support vector machines that depart from the linear functions described above. Such nonlinear extensions transform the given data into higher-dimensional data and use so-called kernel functions. Readers interested in this topic are recommended to consult the book by Christianini and Shawe-Taylor on support vector machines [–CST00].

An example

Let us consider → Table 5.18, which shows data records from a personnel database. The conversion to numerical data has already been carried out. We leave the interpretation of the data to the reader.

Tab. 5.18 Personnel data

Gender	Age	Annual salary	Duration of employment	Position	Educational qualification
0	45	32	10	0	0
1	57	35	25	1	1
0	52	40	5	2	1
0	28	27	6	0	0
0	57	45	25	2	1
1	26	27	8	0	0
0	39	39	4	2	1
0	38	32	3	0	0
0	42	31	15	0	0
1	37	30	10	1	1
0	45	32	8	0	0
1	35	30	15	1	1

We import this data using the CSV reader in KNIME. We then convert the target attribute *educational qualification* into a character string. After dividing the data into training and test sets in a ratio of 70:30, the SVM learner is applied to the training set.

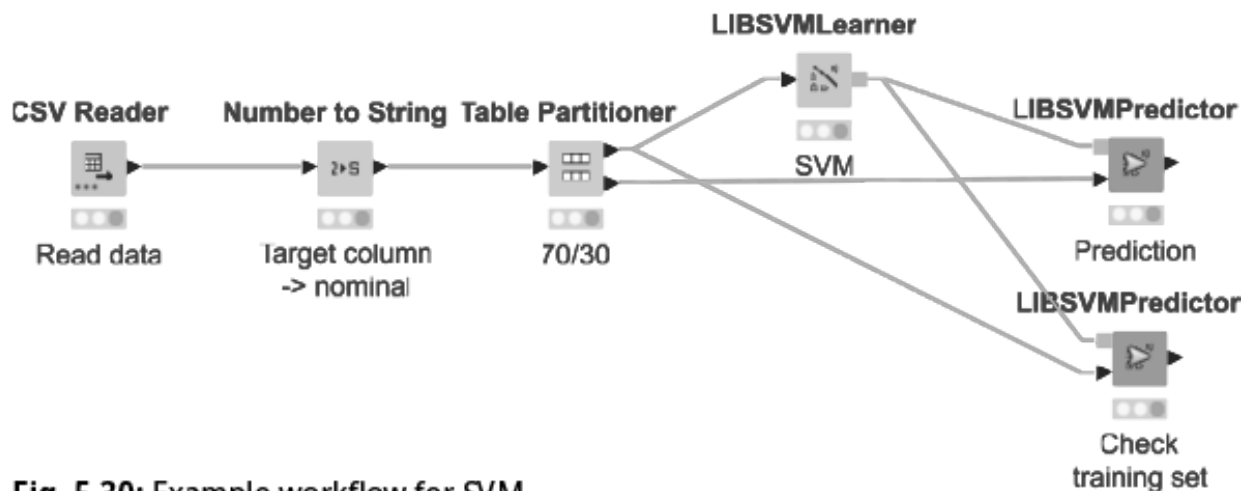


Fig. 5.30: Example workflow for SVM

Fig. 5.30 Example workflow for SVM

The model generated by the SVM now classifies the test set. Although the result is not 100 % correct, the prediction is acceptable, see → Figure 5.31.

File Edit Hilite Navigation View									
Table 'default' - Rows: 4				Spec - Columns: 9		Properties		Flow Variables	
Row ID	I Position	I Gender	I Age	I Annual Inco...	I Length of e...	S Qualificat...	D Prob_0	D Prob_1	S LIBSV...
Row1	1	1	57	35	25	1			0
Row2	2	0	52	40	5	1			0
Row3	0	0	28	27	6	0			1
Row10	0	0	45	32	8	0			1

Fig. 5.31 Prediction generated by SVM on the test data

To check how well the separation has been achieved, the prediction is also applied to the training data. The prediction is perfect on this data, although the probabilities shown in → Figure 5.32 prove that the separation was not completely successful.

File Edit Hilite Navigation View									
Table 'default' - Rows: 8				Spec - Columns: 9		Properties		Flow Variables	
Row ID	I Position	I Gender	I Age	I Annua...	I Lengt...	S Qualificat...	D Prob_0	D Prob_1	S LIBSV...
Row0	0	0	45	32	10	0			1
Row4	2	0	57	45	25	1			0
Row5	0	1	26	27	8	0			1
Row6	2	0	39	39	4	1			0
Row7	0	0	38	32	3	0			1
Row8	0	0	42	31	15	0			1
Row9	1	1	37	30	10	1			0
Row11	1	1	35	30	15	1			0

Fig. 5.32 Prediction generated by SVM on the training data

Task 5.14

Task 5.14 (SVM, weather example).

Find an SVM for the weather example (→ Appendix A.3, numerical variant). Is the game played in the following situations?

outlook = sunny, temperature = 78, humidity = 83, windy = false
 outlook = rainy, temperature = 69, humidity = 94, windy = true

Can you use the data in this form, or does it need to be recoded?

Task 5.15

Task 5.15 (SVM, contact lenses).

Find a supportvector machine for the contactlens data (→ Appendix A.4). Can you use the data in this form, or if not, how does it need to be coded?

Task 5.16

Task 5.16 (SVM, yield data).

Farmers want to improve their harvest collectively.

To this end, the location conditions of the fields were collected for several plant species. Find a support vector machine for predicting the class (*grows*, *does not grow*). Which class does the sample data record fall into (*moist*, *neutral*, 6)?

ID	Moisture	Acidity	Temp. (°C)	Class
1	dry	alkaline	7	grows
2	moist	neutral	8	does not grow
3	dry	acidic	9	grows
4	moist	acidic	5	does not grow
5	dry	neutral	6	grows
6	moist	alkaline	7	does not grow
7	moist	alkaline	7	does not grow
8	moist	neutral	10	grows

5.6 Ensemble learning

Generating good predictions depends on the selection of a good training set (i.e., the subset of the given data used to generate a model) and on the classification method chosen. We address the problem of randomness in classification predictions by taking a democratic approach: we generate several prediction models (neural networks, decision trees, naive Bayes, etc.) and simply make majority decisions for the predictions. We can feed the methods with the same training set or with varying training sets. It is also possible to apply a single method to different training sets.

This approach is referred to as *ensemble learning*. Various models are generated on the basis of specific methods and training sets. A prediction for new data is generated by majority decision. The procedure is illustrated in → Figure 5.33.

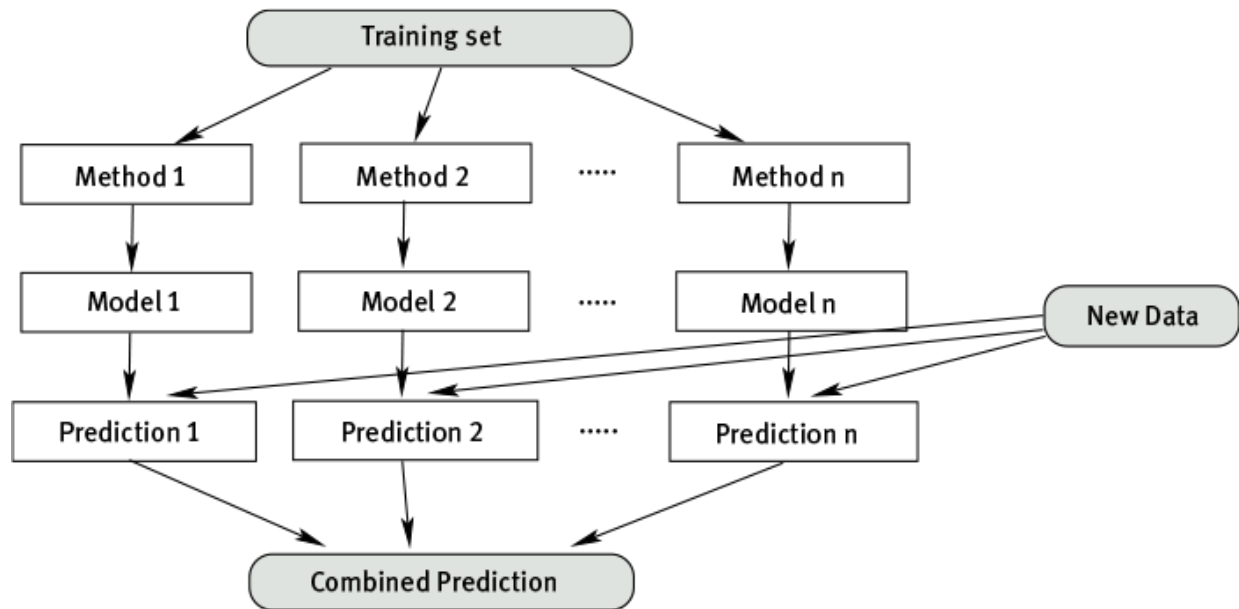


Fig. 5.33 Ensemble learning

Ideally, a majority decision is unanimous. In a classification problem with only two classes, the decision is inevitably made by absolute majority. If several classes are possible, the decision is made by relative majority: the class with the most predictions wins.

There are several forms of ensemble learning, which we will discuss below.

5.6.1 Bagging

The term *bagging* is derived from *bootstrap aggregating*. Bagging was developed by Leo Breiman, who proposed this approach in a research report [→ Bre96]. The basic procedure is illustrated in → Figure 5.34.

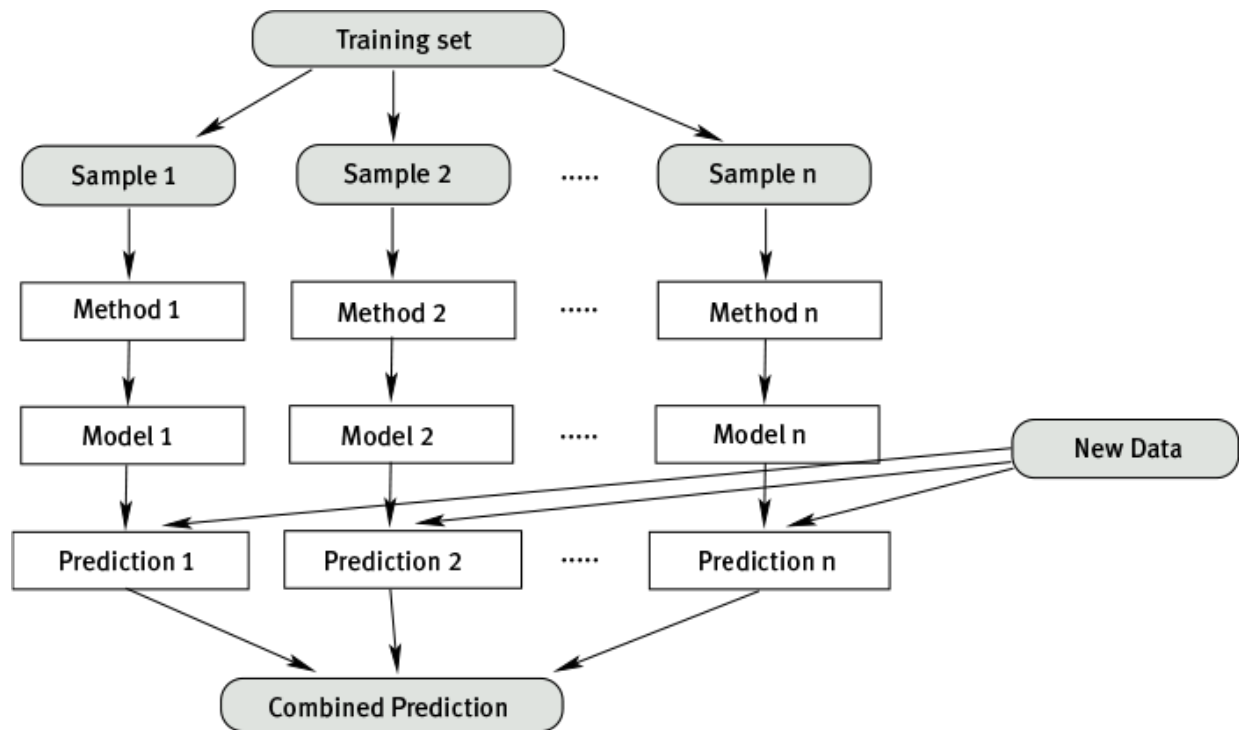


Fig. 5.34 Bagging

Bagging involves working with different training sets. The training sets (samples) are generated using random generators. Multiple selections of data within a training set are permitted. Different methods may be used in bagging.

5.6.2 Boosting

Another ensemble method is *boosting*, see [→Sch90]. Boosting follows the procedure shown in → Figure 5.33, but now iteratively.

1. A classifier K_1 is generated using a random sample.
2. A new sample is generated, but 50 % of the examples that were incorrectly classified in the first run are added. The next classifier K_2 is generated using this data.
3. The third sample contains those examples that were classified differently by K_1 and K_2 . The third classifier K_3 is calculated.
4. Classification is carried out by majority decision using K_1, K_2, K_3 .

The most common implementation of boosting is AdaBoost (adaptive boosting). AdaBoost works with weights. We will refrain from going into mathematical details here and will only present the principle:

- Each data record, i.e., each example, is given a specific weight. Initially, this weight is the same for all data records. This weight determines the probability with which it will be included in the training set. A higher weight increases the probability that the random generator will select this element from the sample set as a training example.
- Boosting works with different classifiers. However, the majority decision is weighted in this case. Classifiers that have demonstrated a high degree of prediction accuracy are given a

higher weight than those that are inaccurate. Naturally, the process starts with equal weights for all classifiers.

The process begins with an arbitrary sample set. In the evaluation of the first run, all data records for which an incorrect result was calculated are given a higher weight. The weights of the correctly classified data records are reduced. This ensures that examples that were incorrectly classified are included in our training set in the next run. At the same time, the weights of the classifiers are adjusted. If a procedure has classified correctly, the weight is increased; otherwise, it is reduced. This process is repeated iteratively.

5.6.3 Random Forest

Random Forest is a form of bagging based on decision trees. This method was also developed by Leo Breiman [→ Bre01].

As with bagging, various classification models (in this case, decision trees) are generated. The training sets can also vary here. Multiple entries of data in a single training set are permitted, as with classic bagging.

In addition, subsets of the available attributes can be formed and decision trees generated that are based solely on these subsets. Classification is again based on majority decision.

6 Cluster analysis

The probability of winning the lottery is slightly greater if you buy a ticket. Yellin's Law

In this chapter, we will look at methods that group objects into suitable sets (*clusters*), see → Sections 3.1 and → 4.6. Cluster analysis is a form of *unsupervised* learning, see → Section 2.6. The aim of cluster analysis is to form groups of *similar* objects. The basic assumption is that objects are more similar the smaller their distance is. We will therefore assume a distance measure that we can use to quantify the similarity of objects. Distance measures are discussed in → Section 2.3.

6.1 Types of cluster analysis

Clustering methods can be divided into four classes:

- Partitioning clustering
- Hierarchical clustering
- Density-based clustering
- Clustering with neural networks

We will not discuss clustering methods that do not fit into these categories, such as graph-based methods.

6.1.1 Partitioning clustering

Partitioning clustering aims to divide the set of objects into k clusters. It begins with an (arbitrary) initial assignment of the objects to the clusters. The clusters can be represented by the *medoid* or *centroid*, see → Section 4.6. From this initial partitioning, the objects are now gradually rearranged between the clusters in

such a way that the quality of the grouping (which we still have to define) continues to improve. To do this, the centroid, for example, is calculated for each cluster in each iteration. All objects are then assigned to the cluster whose representatives they are closest to. These steps are repeated iteratively. The process ends when no more objects are assigned to another cluster, i.e., when the quality of partitioning no longer improves.

Partitioning cluster methods divide the input data into *disjoint* clusters:

- Each cluster consists of at least one object.
- Each object is contained in at most one cluster.

The algorithm is basically as follows.

```
PROCEDURE Partitioning clustering
  Generate (randomly)  $k$  initial clusters  $C_i$ 
    //All objects  $x$  are (randomly)
    assigned to a cluster
    REPEAT
       $swap\_performed := false$ 
    Identify the representatives  $\overline{x_1}, \overline{x_2}, \dots, \overline{x_k}$ 
      of the clusters
    Calculate new assignment of all  $x$  from the
      input data to the clusters  $C_i$ 
    IF an  $x$  is assigned to another cluster THEN
       $swap\_performed := true$ 
    UNTIL NOT  $swap\_performed$ 
  END Partitioning clustering
```

Listing 6.1 (Algorithm for partitioning clustering).

Two methods are considered, both of which belong to the class of centroid methods and assign each object to exactly one cluster, namely:

- k-means (→ Section 6.2) and
- k-medoid (→ Section 6.3).

There are partitioning methods that work on a probability basis and relax the above requirements. Each object can be assigned to different clusters, each with a certain probability (→ Sections 6.4 and → 6.10).

6.1.2 Hierarchical clustering

In hierarchical clustering, a hierarchy of clusters is constructed in such a way that clusters with minimal distance, i.e., greatest similarity, are always merged. For example, higher-level clusters are formed by merging existing clusters.

The process can be represented in a tree structure called a *dendrogram* (→ Figure 6.1).

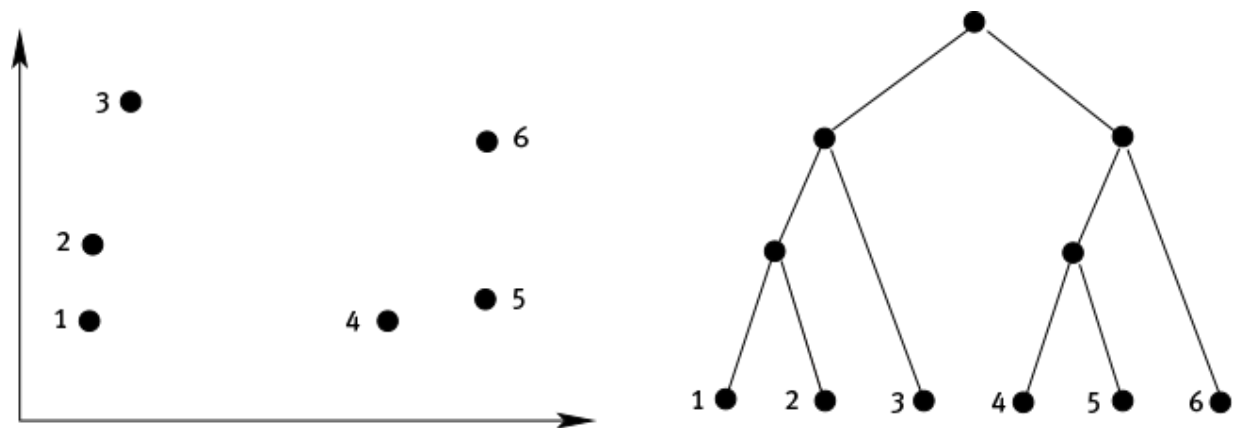


Fig. 6.1 Hierarchical clustering

A dendrogram is a tree whose nodes each represent a cluster and which has the following properties:

- The *root* represents the entire data set.
- Each *inner node* represents the union of all objects contained in the subtrees.
- The *leaves* represent individual objects.

Hierarchical cluster analysis is divided into two categories of methods.

Agglomerative clustering

In agglomerative clustering (→ Section 6.5), the process starts with the smallest resolution of clusters, i.e., the individual objects themselves. Initially, each cluster contains exactly *one* object. The two most similar clusters are then merged into a higher-level cluster. This process is repeated until only one cluster remains.

Divisive clustering

Divisive clustering builds a hierarchy in reverse order. It initially treats the entire data set as one large cluster and divides it into smaller, hierarchically lower clusters. This continues until there is exactly one object in each cluster.

Since there is obviously a whole range of clusterings here, from just *one* cluster to n individual clusters, the question remains as to which clustering should be chosen. One option is to abort the agglomerative or divisive algorithm when all clusters exceed or fall below a certain distance or similarity to each other. You can also stop when a certain number of clusters is exceeded or not reached.

The *disadvantage* of both approaches is that once clusters have been formed, early merging or splitting decisions, they cannot be dissolved again. The *advantage* of these methods is that the number of clusters does not have to be specified in advance.

Hierarchical methods are particularly well suited when one is interested in the connections between clusters. Due to the necessary calculations of pairwise distances for all objects, hierarchical methods are poorly scalable and limited to a few thousand elements in practice. Furthermore, hierarchical methods have problems with outliers.

6.1.3 Density-based clustering

Most clustering methods attempt to group together objects that are similar, i.e., that are close to each other. Partitioning methods, such as the *k-means* method presented in → Section 6.2, result in convex clusters. These methods therefore fail when the objects do not form convex groups, but are arranged as shown in → Figure 6.2, for example.

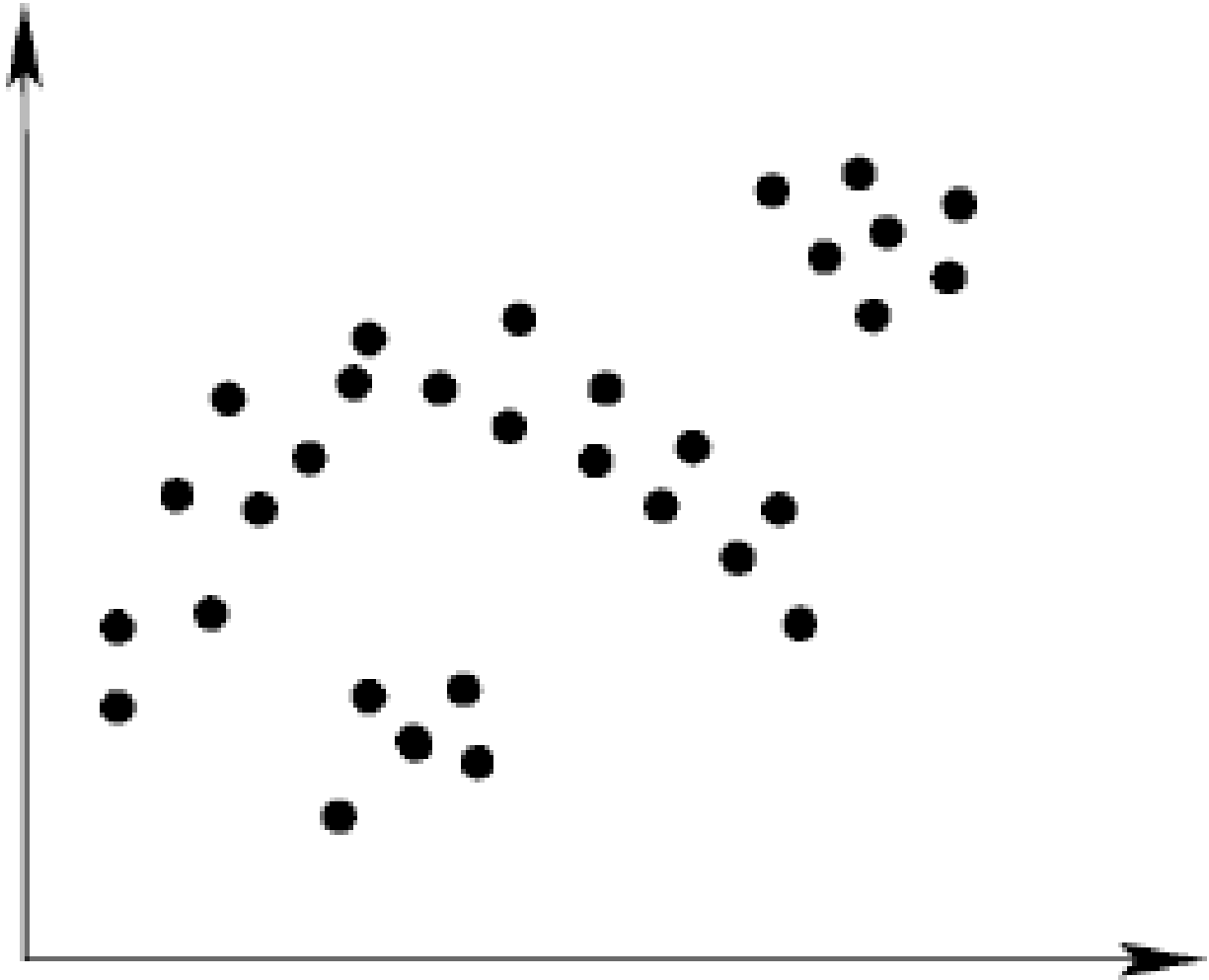


Fig. 6.2 Non-convex groups

We would like to have a clustering as shown in → Figure 6.3.
However, we will not achieve this using the k-means method.

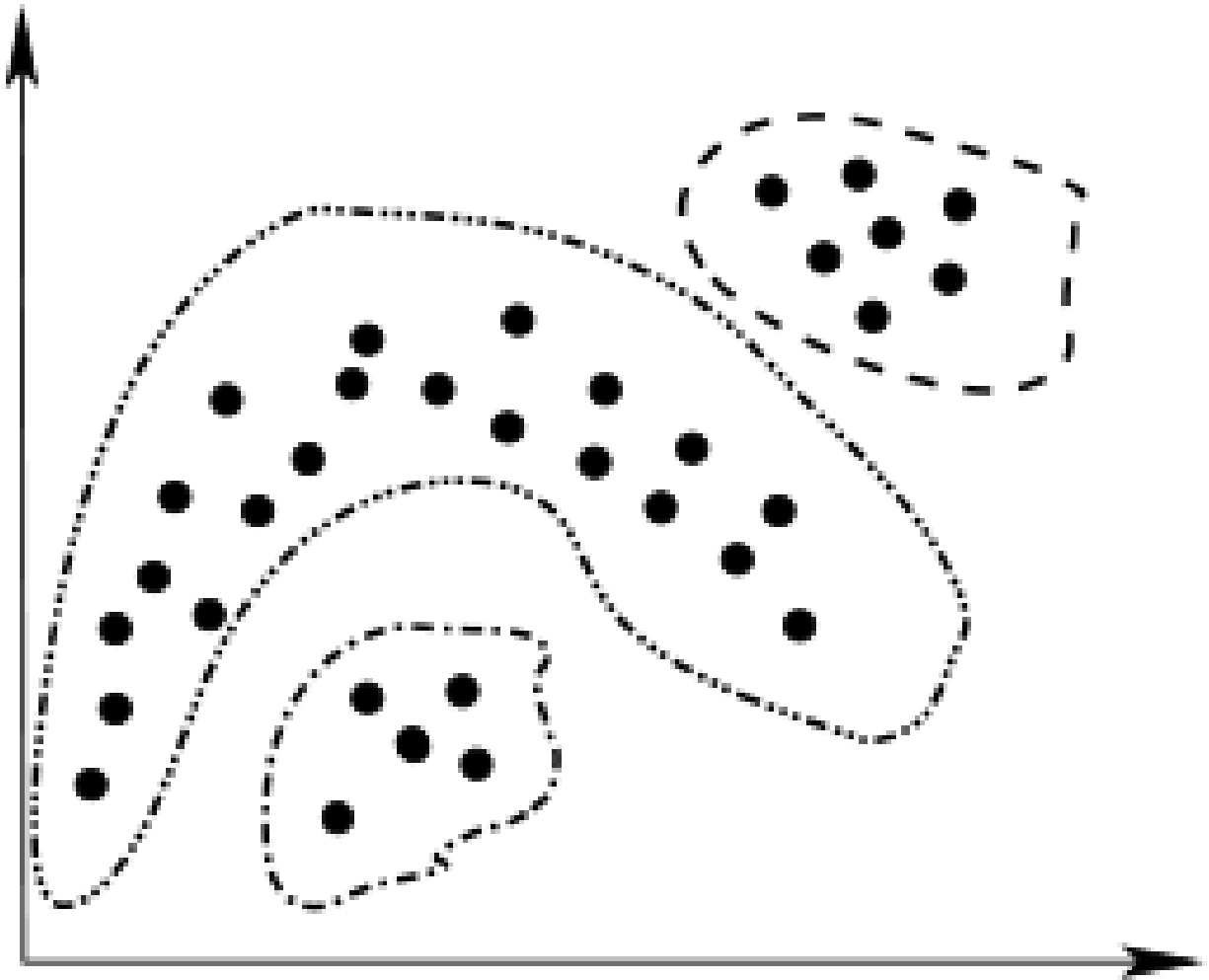


Fig. 6.3 Clustering on non-convex groups

This is what density-based methods are used for (→ Section 6.6). The basic idea is to create clusters so that within *one* cluster, there is a minimum number of other objects in the vicinity of each object. The density of the points within a cluster must therefore exceed a certain threshold value.

6.1.4 Cluster analysis with neural networks

When neural networks are used for cluster formation (→ Sections 6.7 to → 6.9), these neural networks are trained without *supervision*. *Unsupervised learning* methods use *competitive learning*.

A neuron in the neural network wins the competition according to the question: Which neuron in the network is most similar to the input? All input neurons are connected to all other neurons. Each neuron thus has exactly as many incoming connections as there are input neurons. The vector W_j of the incoming connections of a neuron j thus has the same dimension as the input X . Using a similarity measure, the distances of all neurons to the input are calculated, and the neuron with the smallest distance wins. Depending on the method, the weights at the incoming connections of the neuron or even the neighboring neurons are then changed. This results in areas of neurons with similar levels of excitation for similar inputs. These areas then represent the respective clusters.

6.2 The k-means algorithm

The *k-means algorithm* is a popular partitioning method for cluster analysis. The term *k-means* was coined by MacQueen [→ Mac67].

In the k-means method, the number of clusters to be found is given. The centers of the clusters are initialized randomly and then changed iteratively. A cluster is represented by its *centroid* (center of mass).

We will first consider a simple variant of the k-means algorithm.
→ Figures 6.4 to → 6.7 illustrate the procedure.

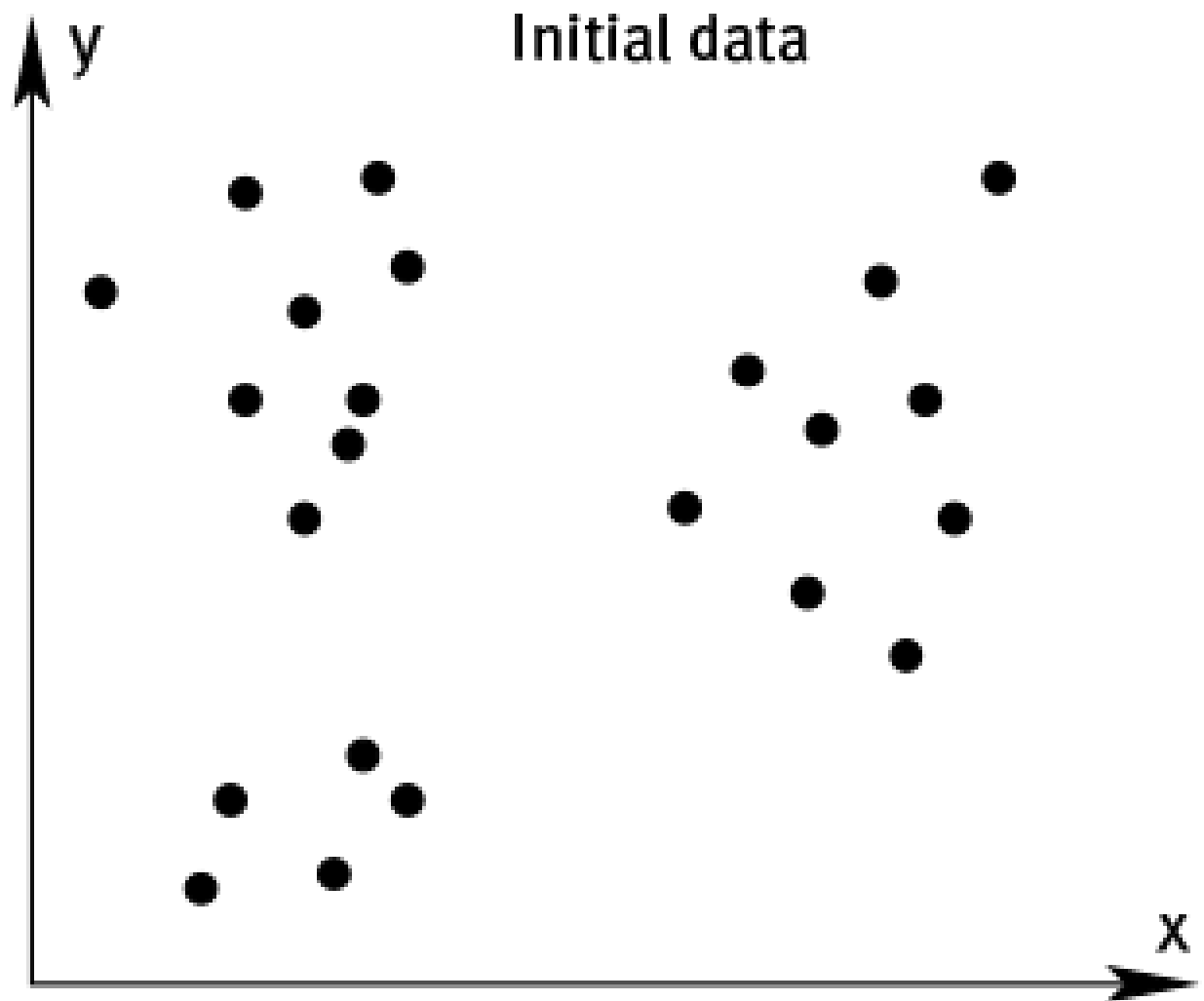


Fig. 6.4 Clustering with k-means, initial data

First, initial clusters are generated, see → Figure 6.5. A random generator can be used for this purpose. In our example, the initial clusters were generated in such a way that they can be easily represented. In step 2, the centroids are calculated, represented by X (→ Figure 6.5). The centroid of a cluster is calculated as the mean value of the respective attribute values of all cluster objects. In two-dimensional real space, the centroid is calculated using the respective mean values of the x and y coordinates of all objects.

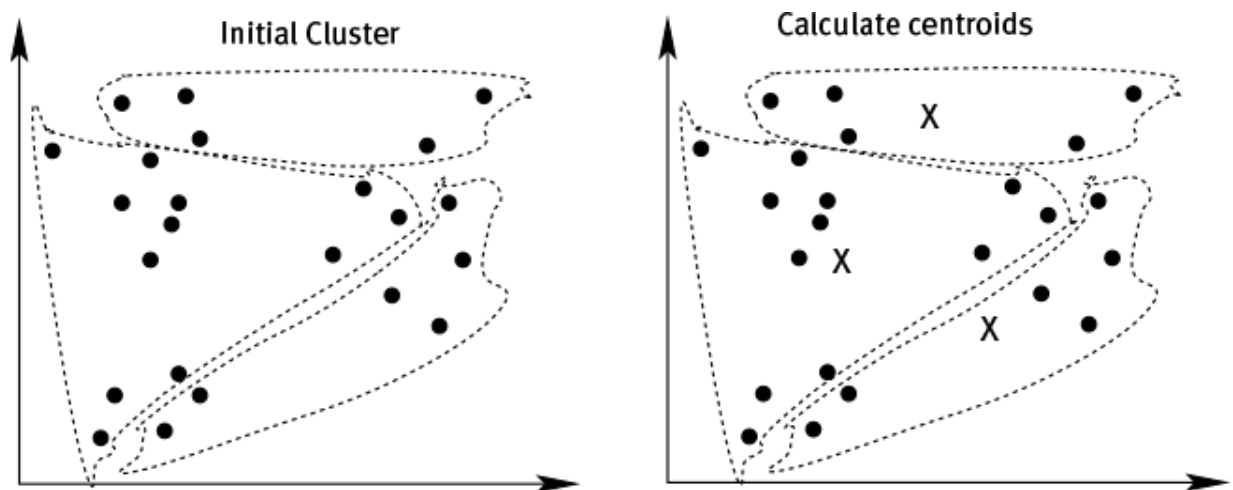


Fig. 6.5 Clustering with k-means, steps 1 and 2

The next step involves reassigning all points to the clusters. To do this, the distances of all points to the three centroids, each representing a cluster, are calculated. Euclidean distance is used as the distance measure.

The recalculation of the centroids, the old cluster centroids are represented by x , the new ones by X , shows a significant shift (→ Figure 6.6).

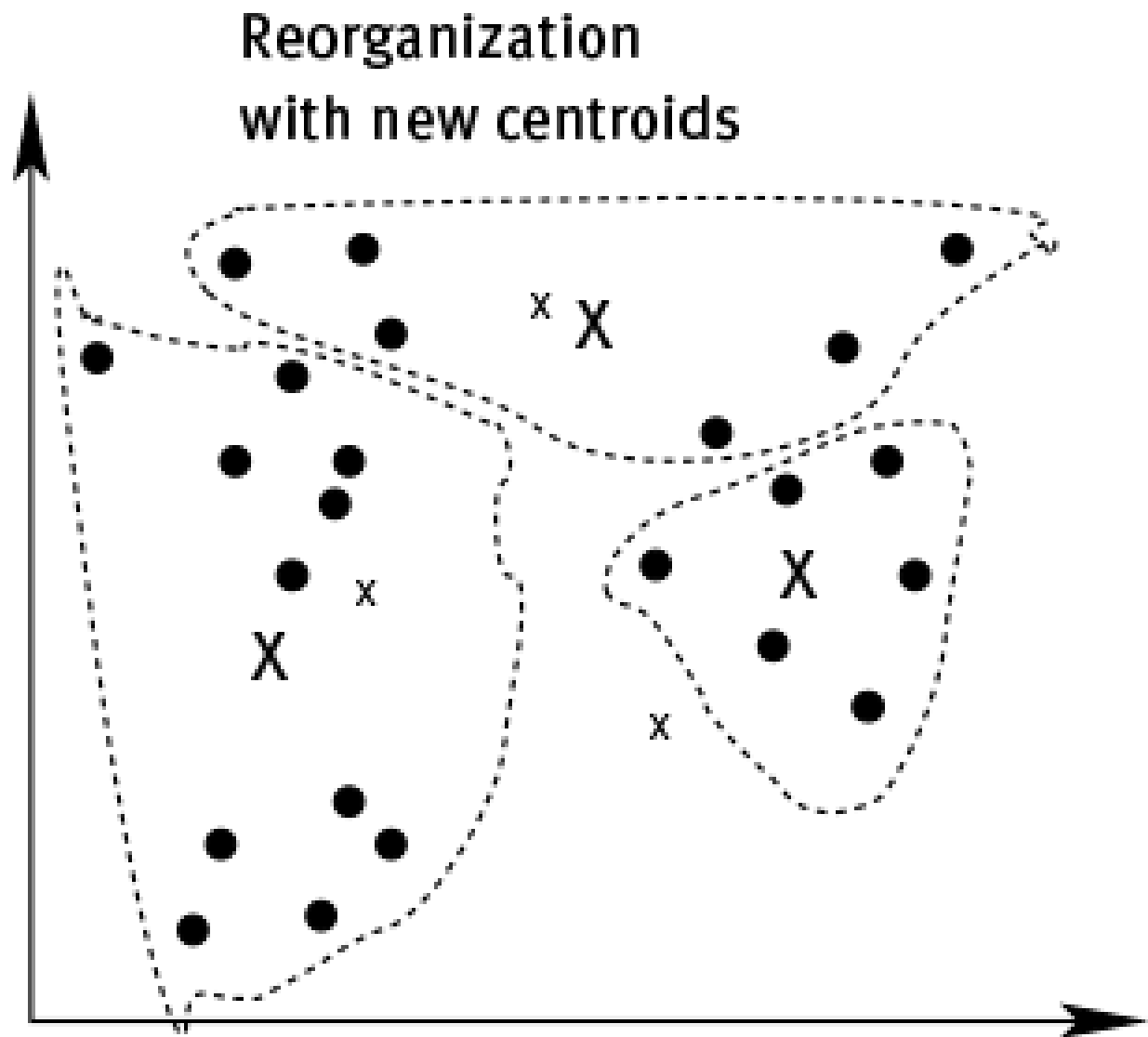


Fig. 6.6 Clustering with k-means, step 3

This process is now repeated iteratively. In each iteration, the nodes are reassigned to the clusters and the centroids are recalculated. For our example, the next steps are shown in → Figure 6.7. The upper cluster gradually takes the upper points from the lower cluster on the left.

The process ends when no more points change to another cluster or when the centroids do not change.

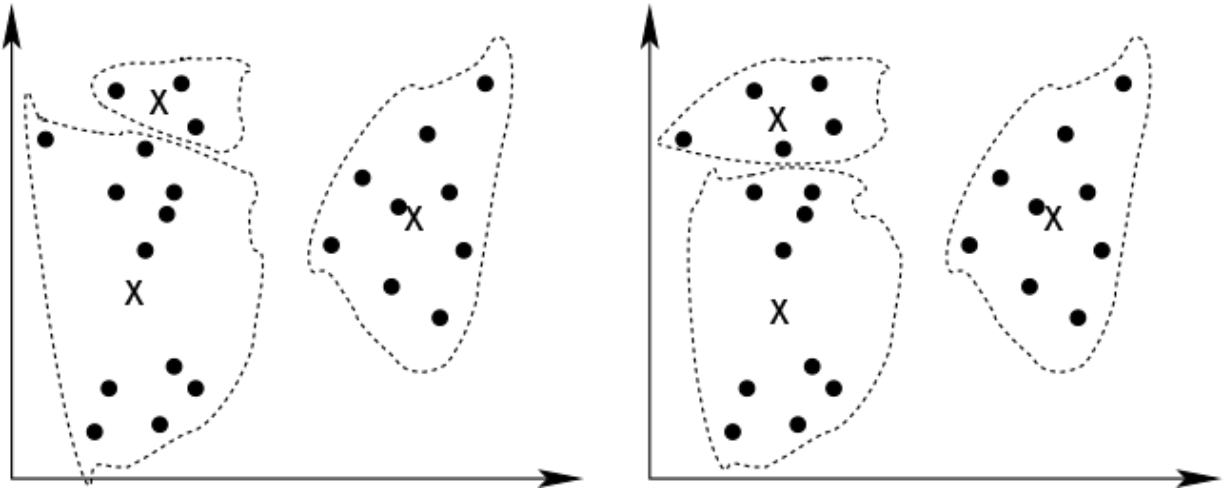


Fig. 6.7 Clustering with k-means, steps 4 and 5

```

PROCEDURE k-means
Generate  $k$  initial clusters  $C_i$  (randomly)
                                //All objects  $x$  are
                                (randomly) assigned to a cluster
                                REPEAT
                                    swap_performed := false
                                    Identify the centroids  $\overline{x}_1, \overline{x}_2, \dots, \overline{x}_k$  of
                                        the clusters
                                    For all  $x$ : Assign  $x$  to the cluster  $C_i$ 
                                        whose centroid  $\overline{x}_i$  is closest to  $x$ 
                                    IF an  $x$  is assigned to another cluster
                                    THEN swap_performed := true
                                    UNTIL NOT swap_performed
                                END k-means

```

Listing 6.2 (k-means, basic version).

What are the *advantages* of the k-means method? One might assume that the method requires many iterations to generate

stable clusters. However, experience shows that this is usually not the case. A stable distribution is often achieved after just a few iterations. The method is easy to implement because, strictly speaking, it consists only of distance calculations and reassignments. In addition, it is quite intuitive and therefore very popular.

One of the *disadvantages* is that the k-means method depends on the quality of the initial partitioning. If the initial partitioning is unfavorable, this can have a negative effect on the result and also on the number of iterations. Finding good initial clusters is therefore very important for this method. The method is sensitive to noise and outliers, as all objects are included in the calculation of the centroids. An outlier pulls the centroids in its direction and can distort a cluster. Furthermore, the effort involved is high, as all distances are recalculated in each step. The runtime can increase significantly, especially with very large amounts of data.

The k-means algorithm only delivers convex clusters. For example, clusters that are located like a tube in n-dimensional space between other clusters cannot be found with k-means. We will discuss finding non-convex clusters in → Section 6.6.

What is the optimal number of clusters? The k-means method does not answer this question. This is up to the user to decide.

The k-means algorithm expects numerical values. Nominal and ordinal attributes must be converted into numerical values.

Despite the disadvantages of this method, the k-means algorithm is a powerful algorithm for cluster analysis.

Let's take a closer look at an example. → Figure 6.8 shows two-dimensional data that we want to divide into clusters. We use the Euclidean distance as distance function.

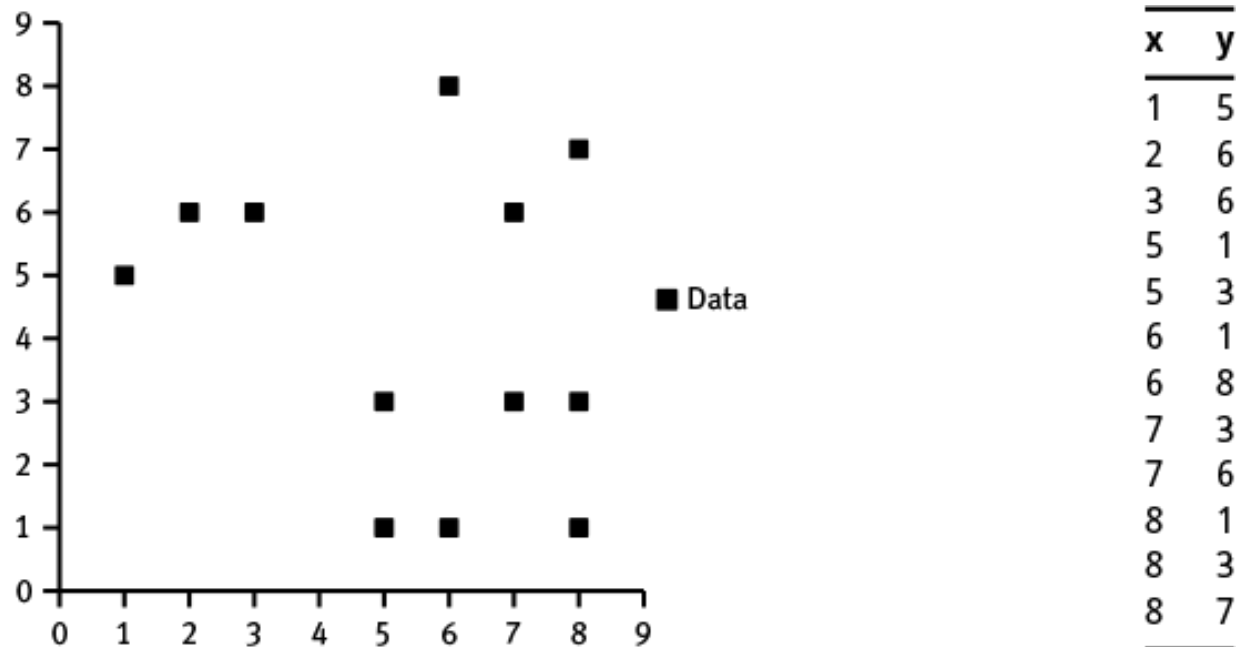


Fig. 6.8 K-means, initial data

We want to form three clusters and therefore need three initial clusters. We arbitrarily set the points (1, 5), (2, 6), and (3, 6) as cluster centers. Then we start the k-means procedure. The initial clusters are poorly chosen, so that the points (1, 5) and (2, 6) initially remain *alone*. All other points are assigned to the cluster (3, 6) because their Euclidean distance to (3, 6) is smaller than to (1, 5) and (2, 6).

In the next step, the point (3, 6) moves to the cluster (2, 6) because its distance to (2, 6) is smaller than to the cluster center (6.3, 3.9). The cluster center of the cluster becomes (2.5, 6) (→ Figure 6.9).

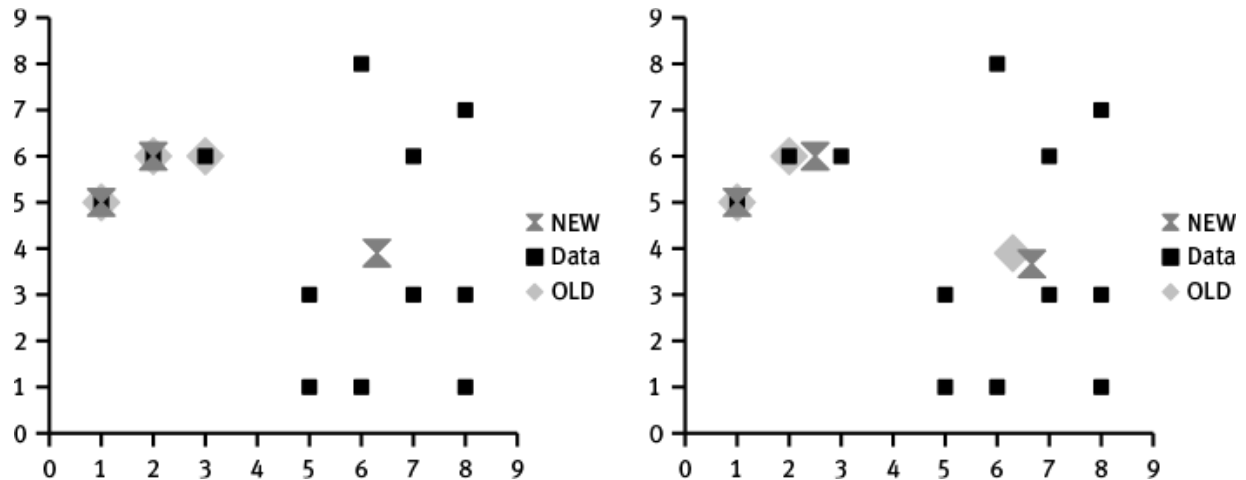


Fig. 6.9 K-means, steps 1 and 2

In step 3, the trend continues with cluster (2.5, 6) absorbing more points from the right cluster. This shifts the centers (→ Figure 6.10).

In step 4, cluster (1, 5) also becomes active. Point (2, 6) is assigned to cluster (1, 5). In step 5, we finally obtain the clusters we expected. In step 6, nothing changes, so the process ends (→ Figure 6.11).

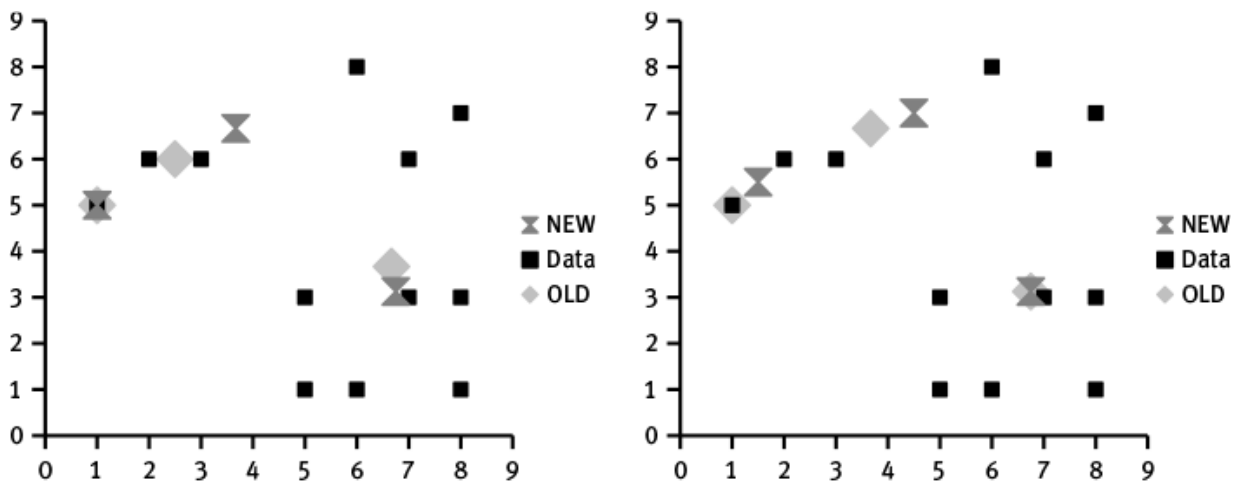


Fig. 6.10 K-means, steps 3 and 4

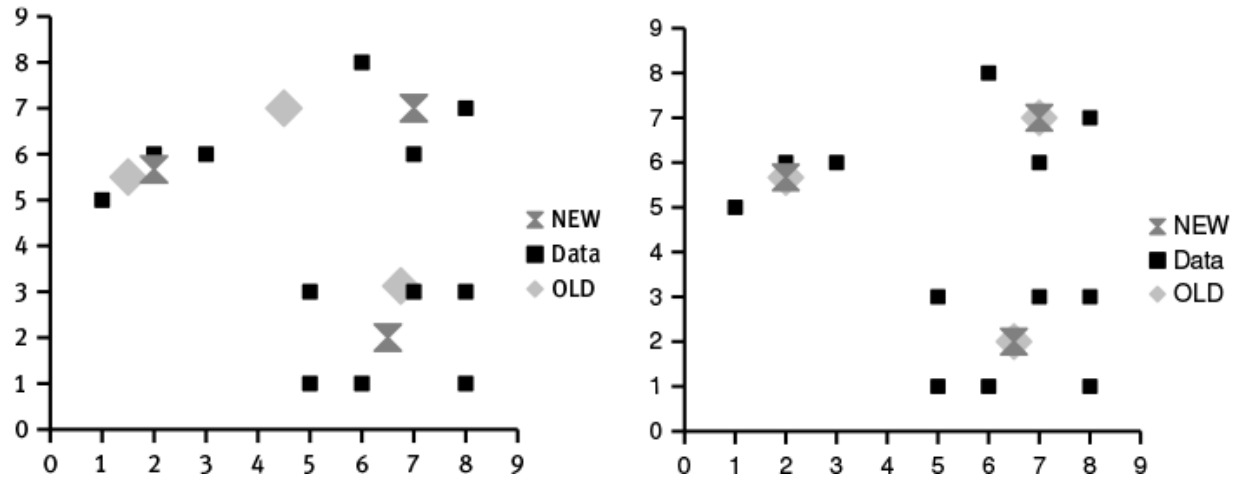


Fig. 6.11 K-means, steps 5 and 6

Let's look at another example. The following data is to be divided into two clusters.

Age	Married	Homeowner	Academic	Income
old	yes	yes	yes	high
old	yes	no	no	low
middle-aged	no	no	no	low
middle-aged	yes	yes	yes	high
young	no	no	no	low
young	yes	no	no	medium
young	yes	yes	yes	medium
old	no	yes	no	high

Can we apply k-means immediately? No, because although we have a distance function in the form of the Hamming distance, we fail when calculating the centers of the clusters, since no arithmetic mean can be calculated for nominal or ordinal values. An alternative is the k-medoid method, see → Section 6.3. However, we will first try to apply the k-means method. K-means requires numerical values, which we do not have. The binary attributes can

be converted to 0 and 1. We note that the attributes *age* and *income* are ordinal attributes. We can convert the values into numbers, taking the order into account:

Age	Income
young → 0	low → 0
middle-aged → 0.5	medium → 0.5
old → 1	high → 1

With this data, k-means can now be applied.

Age	Married	Homeowner	Academic	Income
1	1	1	1	1
1	1	0	0	0
0.5	0	0	0	0
0.5	1	1	1	1
0	0	0	0	0
0	1	0	0	0.5
0	1	1	1	0.5
1	0	1	0	1

The two ordinal attributes *age* and *income* could also be coded using 0, 1, 2. However, this would increase the influence of these two attributes, as the distances between these attribute values become greater. We will discuss this problem in → Chapter 8.

In the k-means method, initialization (the formation of initial clusters) plays an important role. There are several ways to obtain the best possible initial clusters. As already mentioned, it is possible to assign the points to the clusters at random, a very simple approach. Alternatively, good centroids can be determined at the beginning. This can also be done by random selection. However, it

is also possible to select centroids that are as dissimilar as possible, i.e., that are far apart.

The k-means method searches for a partitioning into k clusters that is optimal in a certain sense: the sum of all distances between the objects and their cluster centers should be minimal, see → Section 9.5.

The so-called cost (compactness) of the cluster C_i is measured as

$$\text{Cost}(C_i) = \sum_{x \in C_i} \text{dist}(\bar{x}_i, x) \quad 6.1$$

The total cost (the sum of all cluster costs) must be minimized:

$$\text{Cost} = \sum_{i=1}^k \text{Cost}(C_i) \quad 6.2$$

Due to the high complexity of the k-means method, in each step, all distances must be recalculated, improvements were sought to achieve more favorable runtime behavior for the method. One simple modification is to adjust the centroids of the new clusters not only after rearranging *all* points, but after *each* rearrangement. Thus, if *one* object is found that must change clusters, this change is made *immediately*, see → Listing 6.3. The centroids now only need to be recalculated for two clusters: cluster C_j , which x is leaving, and cluster C_i , to which x is moving to.

When does the process end? This is usually achieved by stopping after a certain number of attempts in which no object x is found that changes clusters. For small amount of data, all objects are considered again. If no object changes clusters, the clustering is complete.

```

PROCEDURE Modified k-means
Generate k initial clusters  $C_i$  (randomly)
    //All objects  $x$  are (randomly)
    assigned to a cluster
    REPEAT
        swap_performed := false
        Identify the centroids  $\overline{x_1}, \overline{x_2}, \dots, \overline{x_k}$  of
            the clusters
        IF There exists an  $x$  that is closer to
        another cluster center  $x_i$  than to its current cluster  $C_j$ 
            (centroid  $\overline{x_j}$ )
            THEN
                Assign  $x$  to Cluster  $C_i$ 
                swap_performed := true
            END IF
        UNTIL NOT swap_performed
    END Modified k-means

```

Listing 6.3 (k-means, modified).

The advantage of this method is that only a limited number of distances need to be recalculated in each step. The disadvantage is that, compared to the basic variant, the speed depends more on the initial partitioning and the convergence depends more on the order of the selected x .

Example 6.1

(Cluster analysis, Iris data). Let us consider the Iris data (see → Appendix A.1). Three Iris species can be identified here:

- Iris setosa,

- Iris virginica,
- Iris versicolor.

The attributes given are the length and width of the sepal and the length and width of the petal.

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class Iris-setosa, Iris-versicolor, Iris-virginica

5.1 , 3.5 , 1.4 , 0.2 , Iris-setosa

4.9 , 3.0 , 1.4 , 0.2 , Iris-setosa

4.7 , 3.2 , 1.3 , 0.2 , Iris-setosa

4.6 , 3.1 , 1.5 , 0.2 , Iris-setosa

.....

Even though we are dealing with a classification problem here, we would like to apply the k-means algorithm to this data. Of course, we leave out the class attribute and then check whether the clustering achieves the correct classification, i.e., whether the clusters formed by the k-means algorithm match the specified classes in the data set.

→ Figure 6.12 shows the clusters calculated with k-means, for reasons of clarity only with regard to the attributes *sepal length* and *petal length*. The values are first normalized to the value range [0, 1], see → Section 8.2.5, in order to avoid different weighting of the attributes in the distance measure, cf. → Figure 6.13.

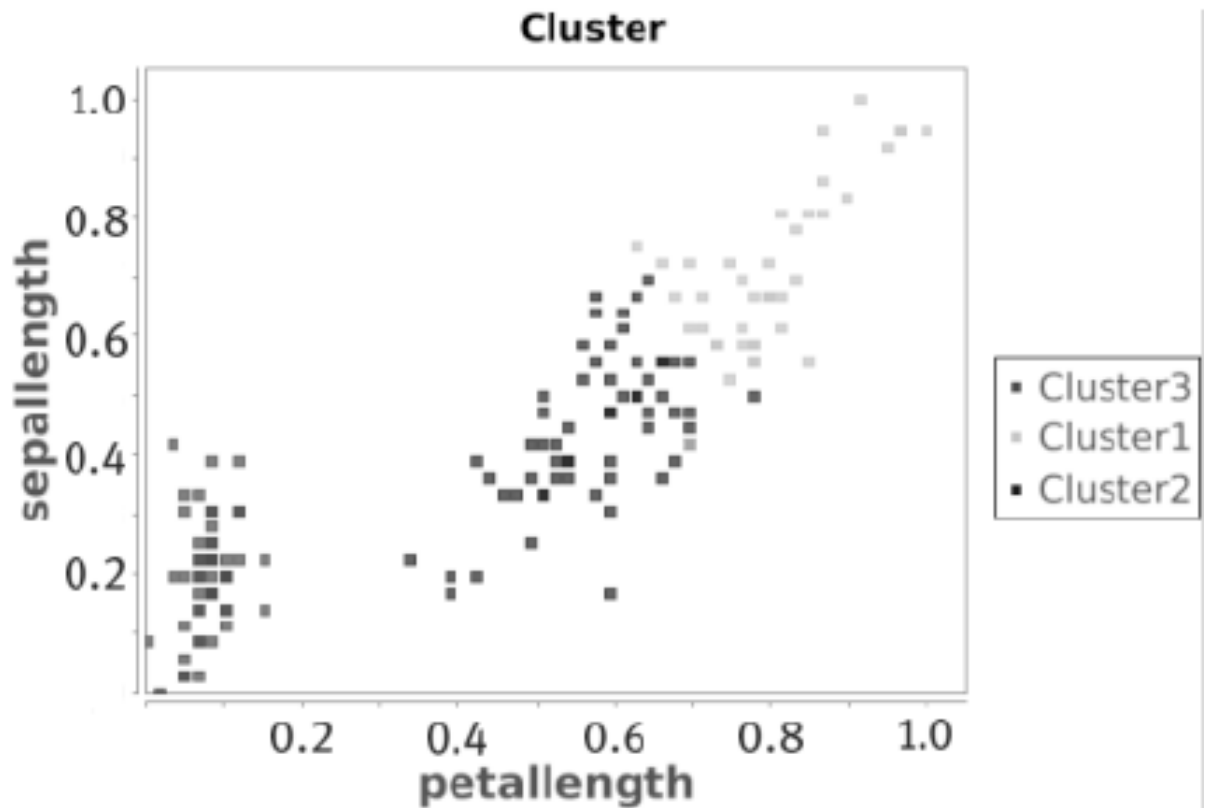


Fig. 6.12 Clustering with k-means on the Iris data

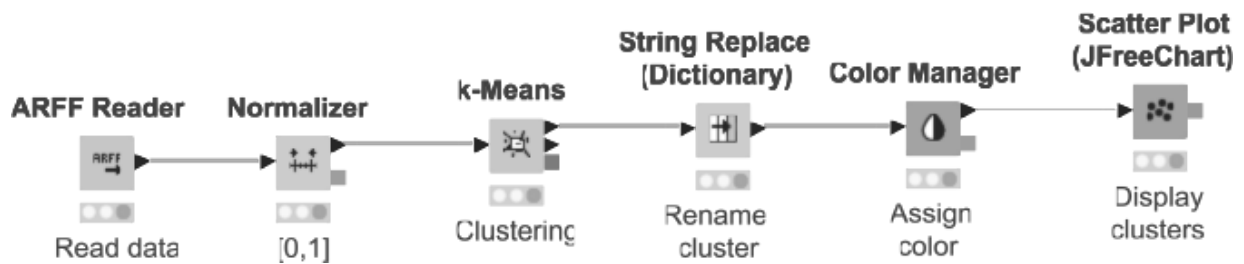


Fig. 6.13 Workflow for clustering Iris data

If we compare these clusters with the original classes (→ Figure 6.14), we see that the clustering only matches completely for one class. Although the other two classes are not recognized 100 % correctly, there are only a few errors. In addition, the original classification also shows some outliers that cannot be recognized by a clustering method.

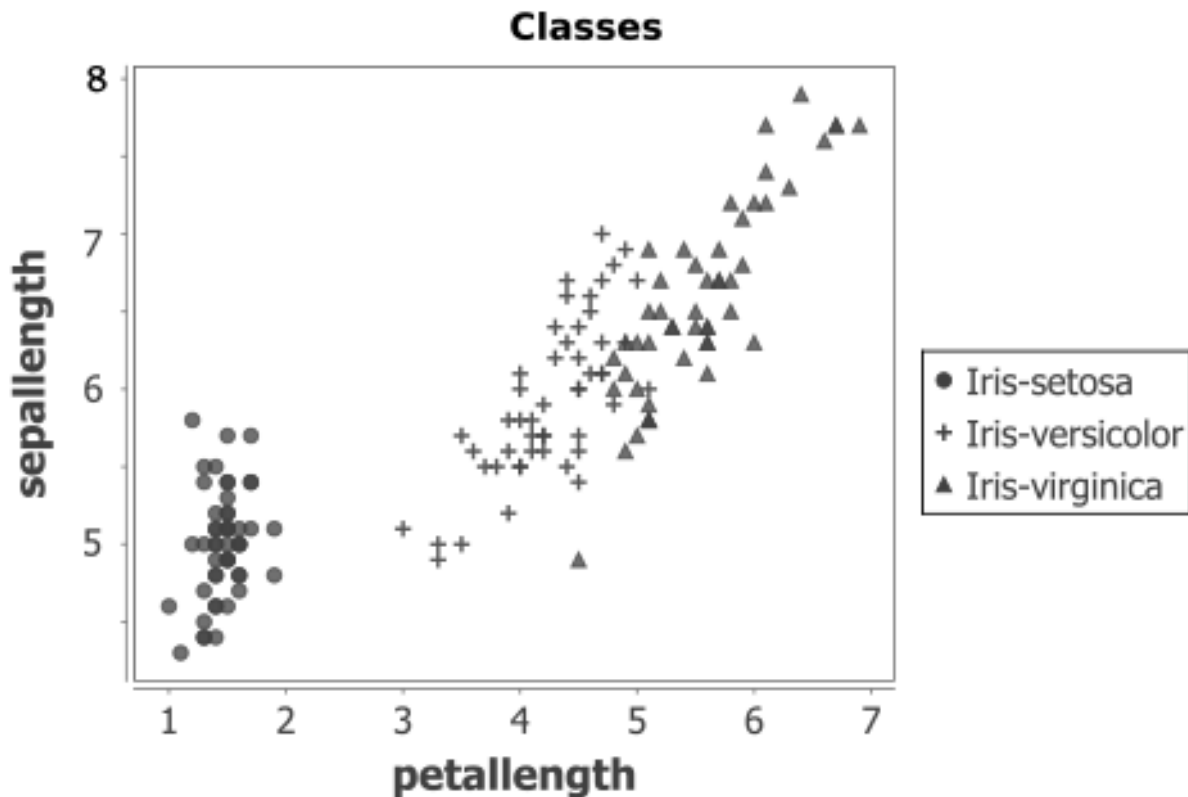


Fig. 6.14 Original classes of the Iris data

We have already pointed out that the k-means algorithm requires metric values, as it is not possible to calculate centroids for nominal and ordinal data. Ordinal data can usually be converted into metric data. Nominal attributes are converted into numerical values by binarization, see → Chapter 8.

There are modifications of the k-means method that allow nominal attributes, for example, the k-modes method.

Task 6.1

Task 6.1 (k-means).

Given the following two-dimensional data records:

x	3	6	8	1	2	2	6	6	7	7	8	8
y	5	2	3	5	4	6	1	8	3	6	1	7

Find three clusters for these data records using k-means! Use the first three data tuples as the initial centers of the clusters. Then calculate the membership of all data records to one of the three clusters. Now recalculate the centroid based on the clusters found. Then reassign all data records to the clusters again, etc. Track the shift of the centroids.

Task 6.2

Task 6.2 (k-means, centroid).

Explain how the k-means method calculates the centroid.

Task 6.3

Task 6.3 (k-means, plant species).

A group of farmers wants to improve their harvest collectively. To this end, the location conditions of the fields of all farmers were collected for several plant species.

Generate two clusters using k-means. Build the clusters without the target attribute *class* and check whether the resulting clusters correspond to the classes. What do you need to do in the data preprocessing to apply the k-means method?

ID	Moisture	Acidity	Temp. (° C)	Class
1	dry	alkaline	7	grows
2	moist	neutral	8	does not grow
3	dry	acidic	9	grows
4	moist	acidic	5	does not grow
5	dry	neutral	6	grows
6	moist	alkaline	7	does not grow
7	moist	alkaline	7	does not grow
8	moist	neutral	10	grows

6.3 The k-medoid algorithm

A method that is very similar to k-means is the k-medoid algorithm. This method uses the so-called *medoid* as the representative of a cluster instead of the centroid. Unlike the centroid, the medoid must be an element of the input data set.

For nominal or ordinal attributes, it is not possible to calculate the centroid: what should the average be between two male and one female person? Therefore, we cannot simply change the k-means algorithm so that the medoid is calculated as the real data record closest to the centroid and the reordering is based on the medoid. Instead, *better* medoids are sought by swapping: in each step, a non-medoid and a medoid swap their current status. The medoid becomes a non-medoid, and the non-medoid becomes a medoid. However, the swap is only executed if the quality of the clustering is improved by the swap, see → Listing 6.4.

```

PROCEDURE k-medoid
  Select  $k$  objects  $m_1 \dots m_k$  as cluster
    representatives
  Assign all objects to a cluster  $m_i$  according to
    the distance function
    REPEAT
      swap_performed := false
      IF There exists an  $x$  (not a medoid) and  $m_i$ 
        (medoid) such that when
          they are swapped ( $x$  becomes
the medoid,  $m_i$  becomes the non-medoid) the cluster
          quality improves
        THEN
          Perform swap and recalculate
            cluster assignment
          swap_performed := true
        END IF
      UNTIL NOT swap_performed
    END k-medoid

```

Listing 6.4 (Medoid-based clustering).

The swap in the THEN branch is actually unnecessary, since the swap has already been executed temporarily to check whether the quality has improved. In the THEN branch, this temporary swap replaces the last valid cluster formation.

The use of the medoid has the advantage that the method is generally less sensitive to outliers than k-means.

The algorithm in → Listing 6.4 only accepts a new medoid if the quality of the clustering improves. The quality of a clustering can again be defined using the costs already introduced:

$$\text{Cost}(C_i) = \sum_{x \in C_i} \text{dist}(\bar{x}_i, x) \quad 6.3$$

The total cost (sum of all cluster costs) must be minimized:

$$\text{Cost} = \sum_{i=1}^k \text{Cost}(C_i) \quad 6.4$$

The algorithm only accepts a new medoid if the total cost is reduced. One of the first k-medoid algorithms was *PAM: Partitioning Around Medoids* [→ KR90]. PAM performs a complete search for the best new medoid.

```

PROCEDURE PAM
  Select k objects  $m_1 \dots m_k$  as cluster
    representatives
  Assign all objects to a cluster  $m_i$  according to
    the distance function
    REPEAT
      swap_performed := false
      cand := {(x, m) | m is medoid, x is a
        non-medoid,
          a swap would result in an
        improvement}
      IF cand  $\neq \emptyset$ 
        THEN
          Find (x, m)  $\in$  cand with the maximum
            improvement
            Swap x and m
            swap_performed := true
        END IF
    UNTIL NOT swap_performed
  END PAM

```

Listing 6.5 (PAM).

The variable *cand* represents the set of all possible candidates for a swap *medoid/nonmedoid*.

The steps before the REPEAT statement are referred to as the *build* phase, and the remaining steps as the *swap* phase. The PAM algorithm calculates a complete matrix in which the distances between the points are stored.

To illustrate this, let's consider the following points in two-dimensional space:

We want to form two clusters. At the beginning, we select the first two points as the medoids of two clusters. We use Euclidean distance as the distance measure. First, the points are assigned to the clusters and the costs are calculated (→ Figure 6.15). The costs add up to 20.9.

No.	x	y
1	2	5
2	4	2
3	5	3
4	1	5
5	3	4
6	3	6
7	5	1
8	6	8
9	7	5
10	7	6

Now PAM swaps a non-medoid with a medoid. If, for example, medoid 2 is swapped with object 8, the new centroids would then be point 1 and point 8, the costs shown in → Figure 6.16 result.

No.	x	y	dist ₁	dist ₂	Cost	Cluster
1	2	5	0	3.61	0	1
2	4	2	3.61	0	0	2
3	5	3	3.61	1.41	1.41	2
4	1	5	1	4.24	1	1
5	3	4	1.41	2.24	1.41	1
6	3	6	1.41	4.12	1.41	1
7	5	1	5	1.41	1.41	2
8	6	8	5	6.32	5	1
9	7	5	5	4.24	4.24	2
10	7	6	5.1	5	5	2
					20.9	

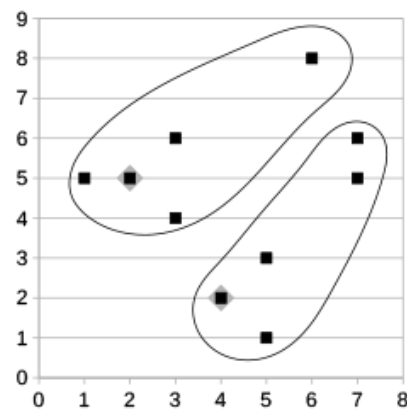


Fig. 6.15 PAM, step 1: assignment

No.	x	y	dist ₁	dist ₂	Cost	Cluster
1	2	5	0	5	0	1
2	4	2	3.61	6.32	3.61	1
3	5	3	3.61	5.1	3.61	1
4	1	5	1	5.83	1	1
5	3	4	1.41	5	1.41	1
6	3	6	1.41	3.61	1.41	1
7	5	1	5	7.07	5	1
8	6	8	5	0	0	2
9	7	5	5	3.16	3.16	2
10	7	6	5.1	2.24	2.24	2
					21.44	

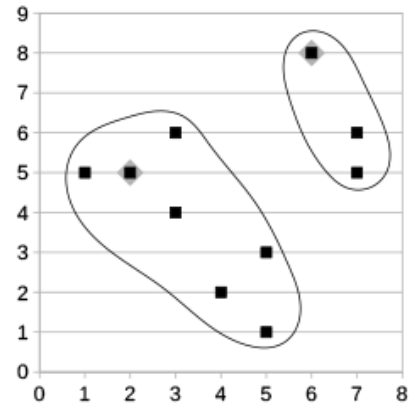


Fig. 6.16 PAM, step 2: medoids 1 and 8

The costs increase, making this exchange unfeasible. However, if point 3 is used as the medoid instead of 2, the costs are lower. The quality increases (→ Figure 6.17).

No.	x	y	dist ₁	dist ₂	Cost	Cluster
1	2	5	0	3.61	0	1
2	4	2	3.61	1.41	1.41	2
3	5	3	3.61	0	0	2
4	1	5	1	4.47	1	1
5	3	4	1.41	2.24	1.41	1
6	3	6	1.41	3.61	1.41	1
7	5	1	5	2	2	2
8	6	8	5	5.1	5	1
9	7	5	5	2.83	2.83	2
10	7	6	5.1	3.61	3.61	2
					18.68	

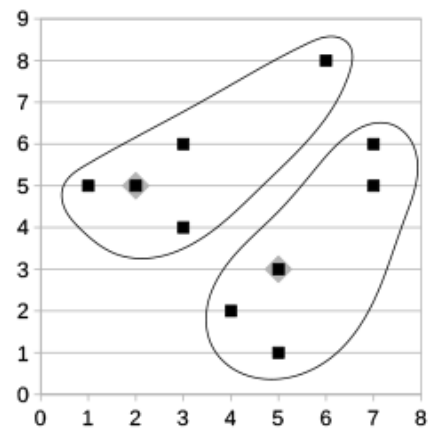


Fig. 6.17 PAM, step 3: medoids 1 and 3

PAM now tries all variants and selects the swap with the greatest improvement. It then searches for the next possible swap.

The PAM algorithm evaluates the quality improvement of clustering for all possible swaps of medoids and non-medoids. Although this usually ensures that good clusters are found, the

runtime increases significantly with the number of input objects, which means that PAM is only suitable for smaller quantities.

A less thorough search is performed by *CLARANS* method (*Clustering Large Applications based on RAN domized Search* [→ NH94], which does not search the entire input space, but only considers a random sample of the data. This makes the method much more efficient without delivering significantly worse results than PAM in practice.

The precursor to CLARANS was CLARA (*Clustering LARge Applications*). CLARA does not work on the entire sample set to form clusters, but selects a subset. The PAM algorithm is then applied to this subset. This method does not necessarily find the optimal clustering, as it is possible that a medoid of the best clustering does not occur at all in the sample set. In order not to be dependent on an unfortunate choice of subset, the entire procedure is therefore repeated several times. The cluster formation that is minimal in terms of the above costs is selected.

While CLARA works with a fixed subset, CLARANS does not adhere to this. The CLARANS algorithm requires two parameters: *numLocal* specifies how often a good clustering is searched for, *maxNeighbors* specifies how many attempts of swapping a medoid with a non-medoid are allowed without an improvement occurring.

PROCEDURE CLARANS

```
    j:=1
    CB := Generate_cluster_formation
    REPEAT
        j:=j+1
        CBj := Generate_cluster_formation
    IF CBj is better than CB THEN CB := CBj
    UNTIL j = numLocal
END CLARANS
```

FUNCTION Generate_cluster_formation

```
    CB := initialize the k medoids
    REPEAT
        i:=0
        Assign each point to its best medoid
        swap_performed := false
    WHILE i < maxNeighbors AND NOT swap_performed
        DO
            Search for random swap partners medoid / non-
            medoid
            IF quality of cluster formation improves
                THEN
                    Perform swap // CB is changed
                    swap_performed := true
                ELSE
                    i:=i+1
                END IF
            END WHILE
        UNTIL NOT swap_performed
    RETURN CB
END Generate_cluster_formation
```

Listing 6.6 (CLARANS).

CLARANS differs from PAM and CLARA in the following ways:

- A maximum of *maxNeighbors* attempts are made to swap medoids and non-medoids in order to find an improvement.
- The *first* replacement that achieves an improvement is carried out.
- The search for *k optimal* medoids is only repeated *numLocal* times.

CLARANS does not aim for completeness, but is faster as a result. The practical complexity is m^3 for PAM and n^2 for CLARANS.

Instead of the centroids or medoids, the *median* can also be used. This leads to the *k-median algorithm*. The difference is that a compromise is sought between the centroid, i.e., a fictitious point that does not necessarily occur in the given data records, and the medoid, a real data record. This compromise means that the median is calculated in each component. The representative does not have to be an existing data record, but each component must have at least one existing value.

If we consider the three data records (0, 4), (2, 0), and (5, 2), we get:

Centroid (2.3, 2)

Medoid (2, 0)

Median (2, 2)

The medoid is an existing data record, while the centroid is a fictitious point. The median may not exist either, but the value in both components occurs in at least one data record.

Task 6.4

Task 6.4 (k-medoid, plant species).

A group of farmers wants to improve their harvest collectively. To this end, the location conditions of the fields of all farmers were collected for several plant species. Generate two clusters using the k-medoid method. Build the clusters without the target attribute *class* and check whether the resulting clusters correspond to the classes *grows* / *does not grow*. What do you need to do in the data preprocessing to be able to apply the k-medoid method?

ID	Moisture	Acidity	Temp. (°C)	Class
1	dry	alkaline	7	grows
2	moist	neutral	8	does not grow
3	dry	acidic	9	grows
4	moist	acidic	5	does not grow
5	dry	neutral	6	grows
6	moist	alkaline	7	does not grow
7	moist	alkaline	7	does not grow
8	moist	neutral	10	grows

6.4 Expectation maximization

Expectation maximization [→ DLR77] belongs to the category of partitioning methods. In contrast to the previous methods, in expectation maximization (EM) clustering, objects are not uniquely assigned to specific clusters. Rather, they can belong to several clusters, each with a certain probability.

A probability distribution is a mapping that assigns a probability to each possible outcome of a random experiment. In clustering by expectation maximization, it is assumed that the data originated from a random experiment, and the clusters are approximated by Gaussian distributions.

A Gaussian distribution is a symmetric probability distribution that induces a bell-shaped graph. Gaussian distributions are used because any other distribution can be approximated by combining

several Gaussian distributions. It is now assumed that the input data originated from a mixture of k *Gaussian distributions*. The goal of the EM algorithm is therefore to find the k Gaussian distributions for which the probability that the given data originated from them is maximum. Similar to the k -means and k -medoid methods, the EM method also starts with arbitrary initial values and iteratively improves the cluster distribution.

The algorithm starts with k randomly selected Gaussian distributions. Now the *probabilities* are calculated with which x (object) originated from one of the k Gaussian distributions C_i ($i = 1, \dots, k$):

$$P\left(x \middle| C_i\right) = \frac{1}{\sqrt{(2\pi)^k |\Sigma_{C_i}|}} \cdot e^{-\frac{1}{2}(x-\mu_{C_i})^T (\Sigma_{C_i})^{-1} (x-\mu_{C_i})} \quad 6.5$$

Here:

- k is the *number of clusters*,
- Σ_{C_i} is the *covariance matrix* for the points in cluster C_i , and
- μ_{C_i} is the vector of the *center* of the cluster i .

Now we calculate the overall probability *density* for x :

$$P(x) = \sum_{i=1}^k W_i \cdot P(x \middle| C_i) \quad 6.6$$

W_i is the number of objects in cluster C_i divided by the total number of objects, i.e., a weighting corresponding to the size of the cluster (relative frequency).

The probability that a given object x belongs to a cluster C_i is (Bayes' theorem, see → Section 5.3.1):

$$P(C_i \middle| x) = W_i \cdot \frac{P(x \middle| C_i)}{P(x)} \quad 6.7$$

To check whether the given data originated from the calculated Gaussian distributions with maximum probability, all probability densities (for all objects) are summed to form a *overall probability density*:

$$E = \sum_x \log(P(x)) \quad 6.8$$

E is to be *maximized*.

1. The iterative algorithm calculates the probabilities $P(x_i)$, $P(x_j | C_i)$ and $P(C_i | x_j)$ for the initial assignments.
2. Then, new mean values of the k clusters are determined from these values.
3. To do this, W_i , μ_{C_i} , and Σ_{C_i} are recalculated.
4. These in turn result in new probabilities.
5. This is repeated until E can no longer be increased.

Mathematically, an object can now belong to several clusters, namely those for which $P(C_i | x_j) > 0$. In order to satisfy the restriction of partitioning clustering that an object belongs to at most one cluster, the objects are assigned to the cluster C_i for which $p(C_i | x_j)$ is maximal.

Like the methods based on center determination, expectation maximization clustering has problems finding clusters with certain properties. For example, clusters with very different spatial structures are difficult to detect using such methods. Density-based clustering (→ Section 6.6) is suitable for such clusters.

6.5 Agglomerative clustering

Agglomerative clustering is a hierarchical clustering method. The algorithm is presented in → Listing 6.7. Again, the procedure is explained using an example.

```

PROCEDURE AgglomerativeClustering
  Create a separate cluster for each object
  // Each object  $x$  from the input data forms its
    own cluster
     $\eta :=$  number of objects
    REPEAT
      Calculate the distances between all clusters
      Merge the clusters  $C_a$  and  $C_b$  with the smallest
        distances
         $\eta := \eta - 1$ 
    UNTIL  $\eta = 1$ 
END AgglomerativeClustering

```

Listing 6.7 (Agglomerative clustering).

Example 6.2

(Agglomerative clustering). In the example shown in
 → Figure 6.1 (p. 140), the individual points $\{1, 2, 3, 4, 5, 6\}$
 from the diagram are located at the lowest level of the
 dendrogram in the first step. These are the *leaves* of the tree.
 In the second step, the two clusters with the smallest
 distance between them are merged into one cluster. Based
 on the Euclidean distance, this means that in our example,
 clusters (points) 1 and 2 now merge into *one* cluster, as they
 have the smallest distance between them. All other
 combinations of points result in a greater distance.

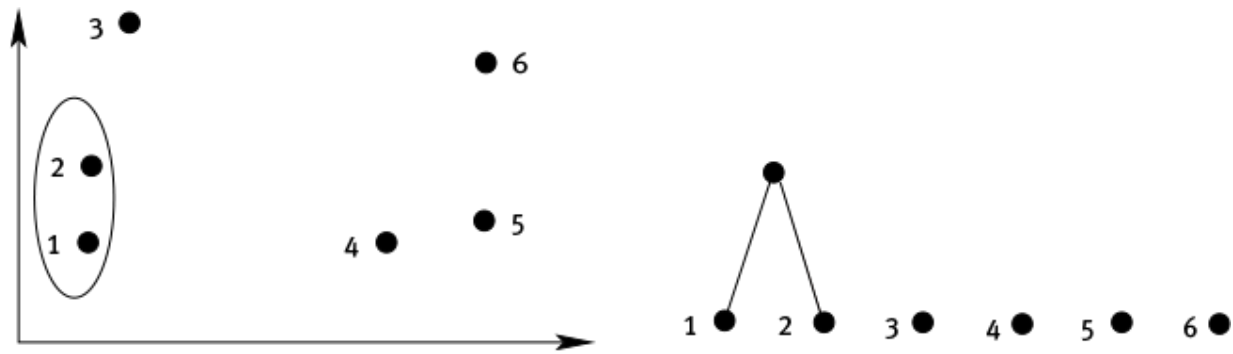


Fig. 6.18 Agglomerative clustering, step 2

Clusters 4 and 5 now merge because they have the smallest distance between them (→ Figure 6.19).

The further behavior depends on the chosen distance measure. If *single linkage* (Equation 6.9) is selected, clusters {1, 2} and 3 merge next, see → Figure 6.20.

In the next step, clusters {4, 5} and 6 are merged, see → Figure 6.21, before a large cluster is created in the final step.

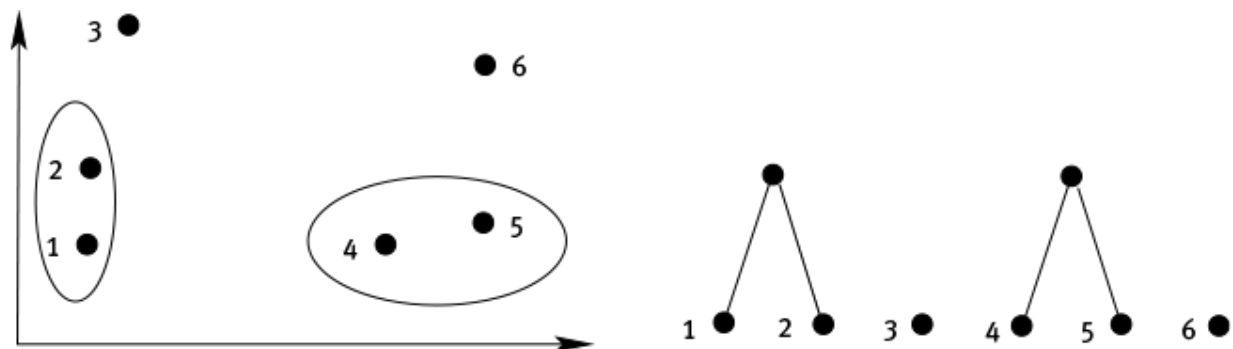


Fig. 6.19 Agglomerative clustering, step 3

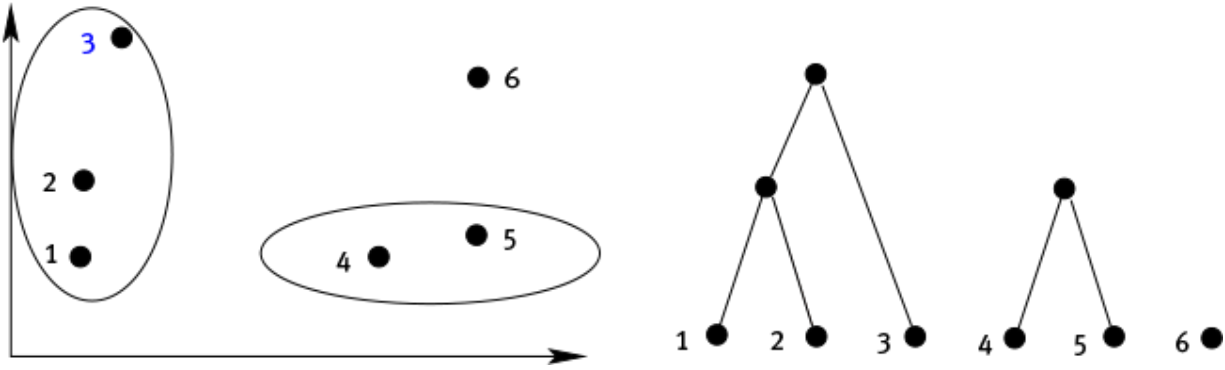


Fig. 6.20 Agglomerative clustering, step 4

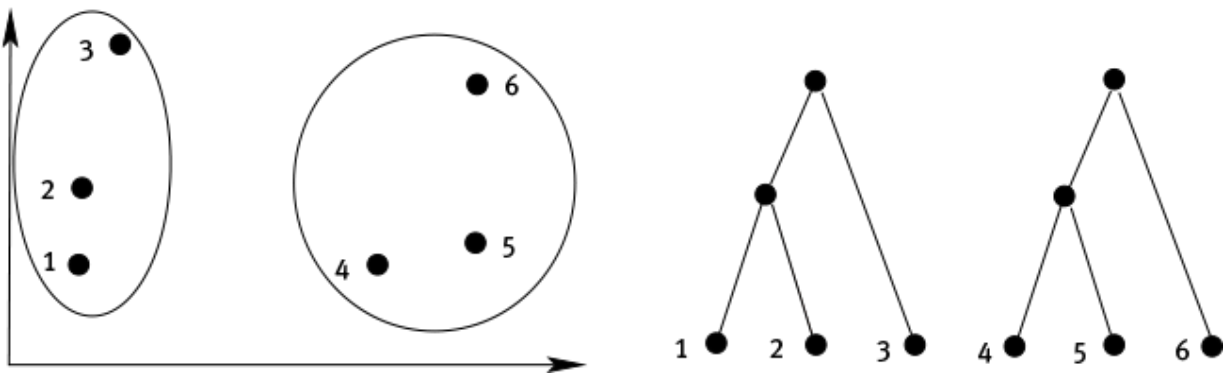


Fig. 6.21 Agglomerative clustering, step 5

There are several approaches for determining the distance between two clusters.

Single linkage (nearest neighbor)

The distance between two clusters C_a and C_b is defined by the smallest distance between two objects $x_a \in C_a$ and $x_b \in C_b$ ($\rightarrow$ Figure 6.22):

$$\text{dist}_{\text{SingleLinkage}}(C_a, C_b) = \min_{x_a \in C_a, x_b \in C_b} \left(\text{dist}(x_a, x_b) \right) \quad 6.9$$

Complete linkage (furthest neighbor)

The distance between two clusters C_a and C_b is defined by the greatest distance between two objects $x_a \in C_a$ and $x_b \in C_b$ ($\rightarrow$ Figure 6.22):

$$\text{dist}_{\text{CompleteLinkage}}(C_a, C_b) = \max_{x_a \in C_a, x_b \in C_b} \left(\text{dist}(x_a, x_b) \right) \quad 6.10$$

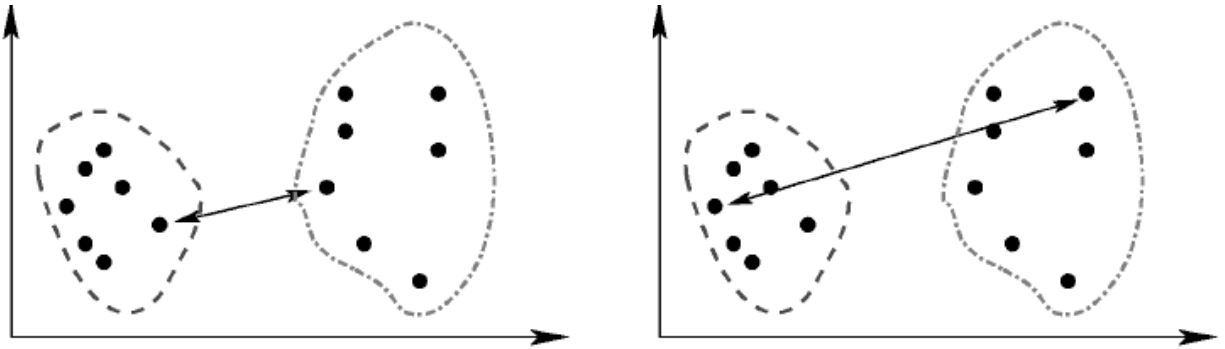


Fig. 6.22 Distance between clusters: single linkage (left) and complete linkage (right)

Average linkage (within groups)

This variant of distance seeks a compromise between *single linkage* and *complete linkage*. The distance between two clusters C_a and C_b is defined by the average distance between objects $x_a \in C_a$ and $x_b \in C_b$ ($\rightarrow$ Figure 6.23). With $|C_a|$ we denote the number of objects located in cluster C_a .

$$\text{dist}_{\text{AverageLinkage}}(C_a, C_b) = \frac{1}{|C_a| \cdot |C_b|} \cdot \sum_{x_a \in C_a, x_b \in C_b} \text{dist}(x_a, x_b) \quad 6.11$$

Centroid

The distance between two clusters C_a and C_b is defined by the distance between their centroids c_a and c_b ($\rightarrow$ Figure 6.23). The

centroids are shown in gray.

$$\text{dist}_{\text{centriod}}(C_a, C_b) = \text{dist}(c_a, c_b) \quad 6.12$$

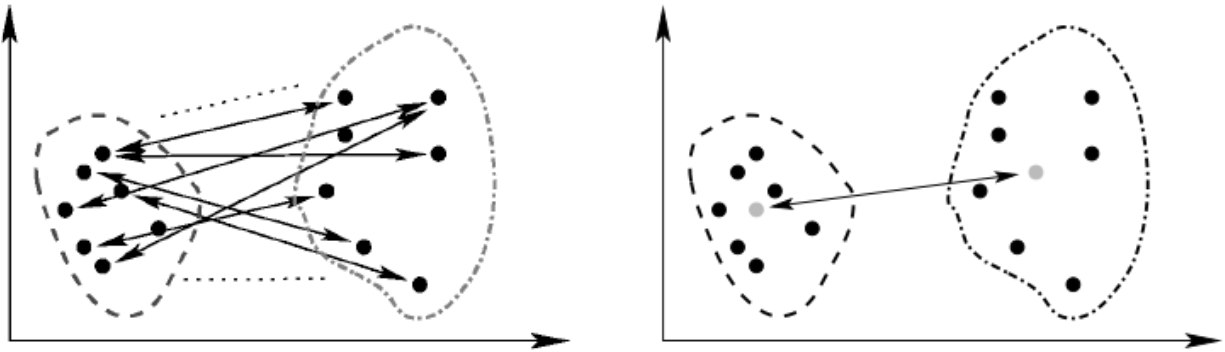


Fig. 6.23 Distance between clusters: average linkage (left) and centroid (right)

Of course, you can also work with the medoids instead of the centroids. The distance between two clusters C_a and C_b is then defined by the distance between their medoids m_a and m_b (→ Figure 6.24):

$$\text{dist}_{\text{Mediod}}(C_a, C_b) = \text{dist}(m_a, m_b) \quad 6.13$$

In → Figure 6.24, the medoids were chosen as the points closest to the centroids.

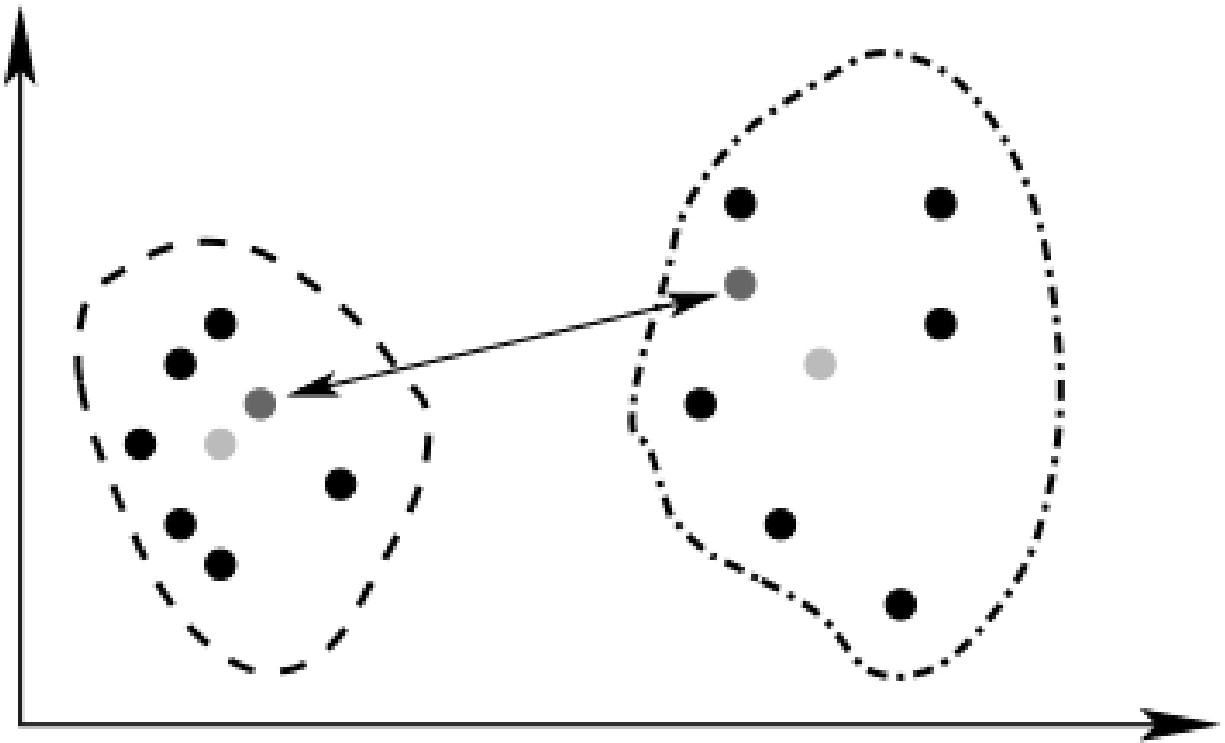


Fig. 6.24 Distance between clusters: medoid

Ward

This method (see [→ War63]) is based on the variance within a cluster. The variances of all clusters are calculated, and then the two clusters are selected whose union results in the minimum increase in the sum of all variances.

In addition to the variants presented here, there are other ways to measure the distances between clusters, such as the median-based method.

We can conclude that there is no *single* method that delivers the best results for all applications. Consequently, you will have to experiment with several distance measures.

6.6 Density-based clustering

Clusters with widely differing spatial structures are not recognized by either partitioning or hierarchical clustering. In such cases, *density-based clustering* is advantageous.

So far, we have considered a cluster to be a set of objects that are as close as possible to a certain center point. Alternatively, a cluster can also be viewed as a set of objects that have a certain *density* among each other and are separated from other clusters by regions of lower density. The local point density for each point in the cluster must reach a predetermined minimum density. The set of objects that make up a cluster is spatially connected. Unlike the k-means and k-medoid methods, however, this set does not have to be convex.

The starting point is therefore that we want to achieve a minimum density. This density is usually specified by two parameters:

- a radius $\epsilon > 0$ and
- the parameter *minimumPoints*, which specifies the minimum number of objects that must be in the vicinity of a point.

A classic example of density-based clustering is the *DBSCAN* algorithm (*Density-Based Spatial Clustering of Applications with Noise*) [Est+96]. It requires the given data set *DB* and the two density parameters ϵ and *minimumPoints*.

```

PROCEDURE DBSCAN( $DB$ ,  $\epsilon$ , MinimumPoints)
    clusterID := 0
    Label all  $x \in DB$  as unclassified
    FORALL  $x \in DB$ 
        IF  $x$  is unclassified THEN
            IF expand( $DB$ ,  $x$ , clusterID,
                minimumPoints,  $\epsilon$ )
                THEN clusterID := clusterID + 1
    END DBSCAN

```

```

PROCEDURE expand( $DB$ ,  $x$ , clusterID, minimumPoints,  $\epsilon$ )
     $Set = \{y \in DB \mid dist(x, y) < \epsilon\}$ 
    IF not enough objects in  $Set$ , i.e.  $|Set| <$ 
        minimumPoints,
        THEN label  $x$  as noise and RETURN false
    FORALL  $y \in Set$ : label  $y$  with the current clusterID
    Delete  $x$  from  $Set$ 
    FORALL  $z \in Set$ 
         $Set_2 = \{y \in DB \mid dist(z, y) < \epsilon\}$ 
    IF sufficient objects in  $Set_2$ , i.e.  $|Set_2| >$ 
        minimumPoints THEN
        FORALL  $s \in Set_2$ 
            IF  $s$  does not belong to any cluster (unclassified
                or noise) THEN
                IF  $s$  unclassified THEN Add  $s$  to  $Set$ 
                Label  $s$  with clusterID
                Delete  $z$  from  $Set$ 
            RETURN true
    END expand

```

Listing 6.8 (DBSCAN algorithm.¹.

At the beginning, the algorithm initially labels all objects as *unclassified*. Then, an arbitrary object from the set is selected and the set of objects in the ϵ neighborhood is calculated. This already happens in the Boolean function *expand*. If the number of neighboring objects is too small, the current object is labeled as *noise*. If there are enough objects, then a new cluster is created. Now all that remains to be done is to see whether the cluster can be expanded further along the new objects, i.e., the neighbors of the current object. In doing so, it is only permissible to go beyond the neighboring points if these neighboring points also have enough neighbors in their neighborhood.

DBSCAN recognizes three types of points:

1. Points that are *dense* themselves, called *core objects*. These are points in whose vicinity the minimum number of dense neighboring points is reached.
2. Points that are *density-reachable*. These are objects that are reached by a core object of the cluster but are not core objects themselves. In other words, they do not have enough *dense* neighboring points, but are themselves within the catchment area of a core object. These points form the edge of the cluster.
3. *Noise points*, which are neither dense nor reachable by density. These are points that neither have enough *dense* neighboring points nor are within the catchment area of a core object. In the algorithm given in → Listing 6.8, these are labeled with *noise*.

The DBSCAN procedure can be easily illustrated. Let's consider a set of points in two-dimensional space and start with an arbitrary point. We require a minimum of two neighboring points and specify a corresponding radius. This is shown in → Figure 6.25. For clarity, we have chosen a point at the edge.

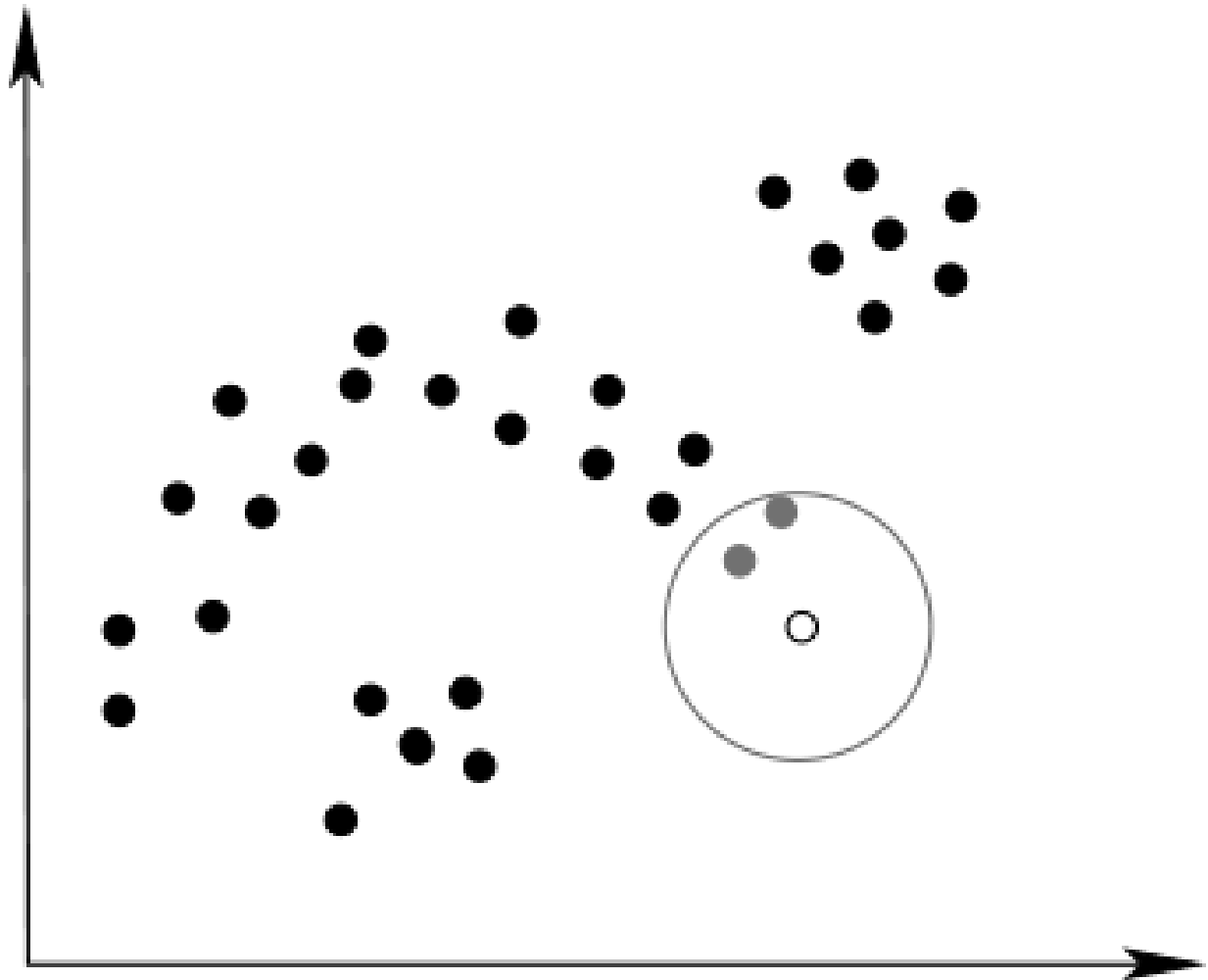


Fig. 6.25 DBSCAN, step 1

The gray points lie within the required proximity to the starting point, using the Euclidean distance. This means that our point is a core object. A new cluster is created. The process now continues with the gray points. This is the Set.

For each gray point, the Set_2 of the closest neighbors is now calculated (→ Figure 6.26, left).

Both points are core points themselves. Therefore, the process now continues with the next points. The points from the curved point cloud are successively reached and form a cluster (→ Figure 6.26, right).

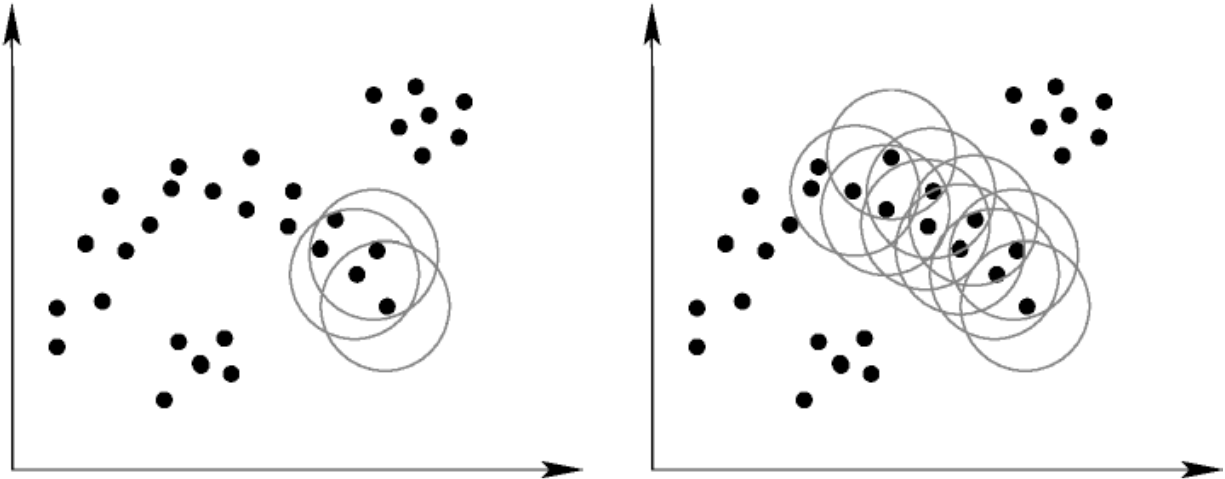


Fig. 6.26 DBSCAN, step 2 and further steps

Since the algorithm runs through all points, the other points are also processed, so that the three desired clusters are found by DBSCAN. However, there may be points that are not assigned to any cluster. These are the noise points. It is also possible that points are reachable from several clusters. In this case, the assignment is made according to the algorithm in → Listing 6.8 to the cluster that reaches this point last. However, neither of these cases applies in our example.

One advantage of DBSCAN is that the number of clusters is determined automatically. Problems arise with DBSCAN when it comes to choosing the minimum number of points and the neighborhood radius ϵ . Both parameters must be specified for the algorithm. This again provides scope for experimentation.

There are a number of extensions to the DBSCAN algorithm, of which the DENCLUE algorithm and the OPTICS algorithm (Ordering Points To Identify the Clustering Structure) are representative examples.

Artificial neural networks can also form clusters that have a non-convex shape.

6.7 Cluster formation using self-organizing maps

Artificial neural networks are used for several application classes of data mining. Depending on the task, a special network architecture and tailored learning methods are required. *Self-organizing maps* (SOM), also known as *Kohonen networks* or *Kohonen feature maps*, are capable of clustering data.

6.7.1 Structure

A self-organizing map consists of two layers of neurons, see → Figure 6.27:

1. The input layer is one-dimensional and represents the data. It is the interface to the outside world: the data to be analyzed is applied to the input neurons.
2. The map or Kohonen layer. This is both the processing and *output layer*.

The input layer is fully connected to the map layer. The neurons in the map layer are all implicitly connected to each other: there are no trainable weights; the geometric distance between two neurons is considered to be the weight of the connection between the neurons. The proximity or distance of one neuron to another is then taken into account in the training process: the closer the proximity, the stronger the influence on the connections of a neighboring neuron to the input neurons.

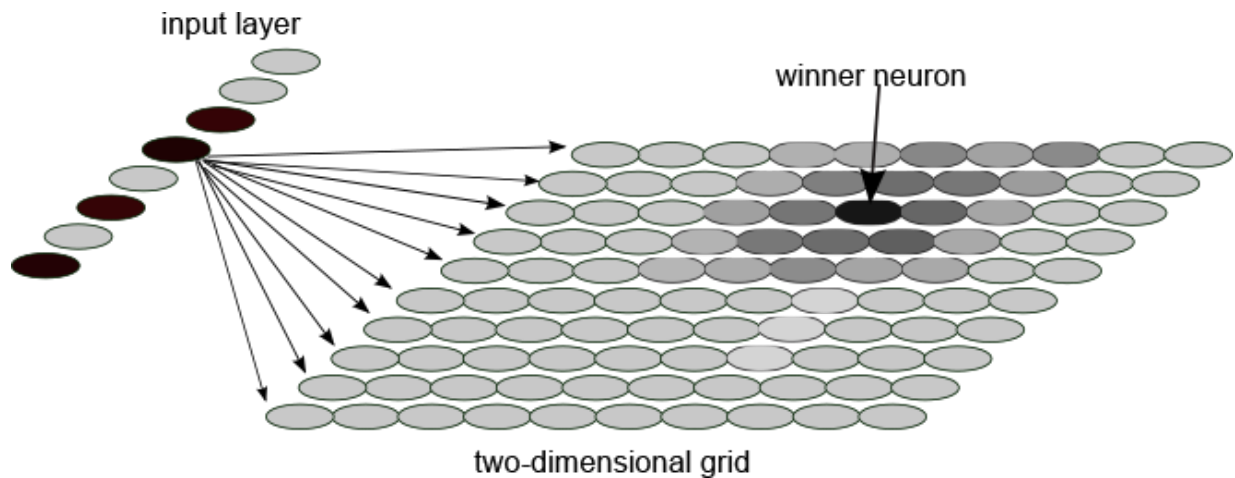


Fig. 6.27 A self-organizing map with a 2-dimensional grid

If you want to develop a self-organizing map for a clustering problem, you need to determine the sizes of the two layers. The number of neurons in the input layer corresponds to the number of values in a pattern. All data must be in the value range $[0, 1]$; preprocessing, which may increase the number of input neurons, may be necessary.

The size of the map layer cannot be determined exactly. Experiments with several maps and different dimensions are necessary here. A good starting point is a square map with a size of 10×10 neurons.

6.7.2 Learning

Learning a self-organizing map is *unsupervised learning*: there is no training data available that could be used to check the quality of the output of a self-organizing map. Thus, the self-organizing map is trained solely by creating the input data. The aim of the training is to create clusters. The starting point here is that the Kohonen layer is not considered solely as a set of neurons, but rather as a two-dimensional arrangement of neurons in which the positions of the neurons play a decisive role. Similar data records, patterns, are mapped to similar areas of the self-organizing map. These areas

can then be regarded as clusters. There is no clear representative of a cluster; instead, several neurons in an area can perform this function.

How is a pattern assigned to a cluster? Since each neuron in the input layer is connected to each neuron in the map layer, the network input of a neuron consists of all outputs of the input neurons and a corresponding number of weights at the connections, see → Figure 6.28.

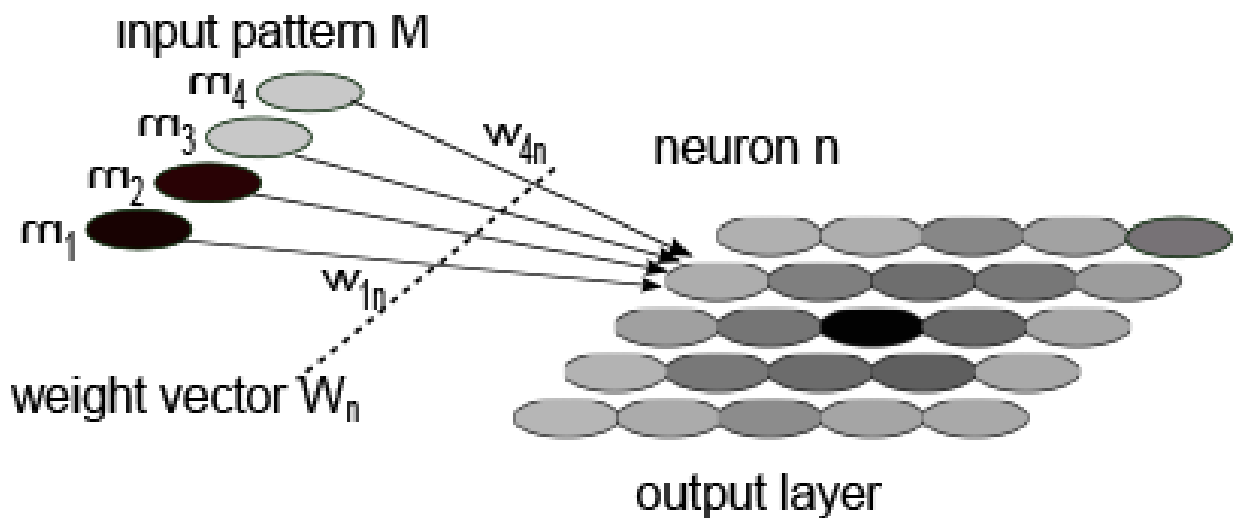


Fig. 6.28 Input of a neuron in the 2-dimensional map layer: $M \times W_n$

For each neuron in the map layer, we are dealing with two vectors:

1. The input pattern is a vector M consisting of k values.
2. The set of all connection weights w_{in} to a neuron n form a vector W_n which determines the activation of the neuron n .

During processing, a pattern is randomly created and the neuron determines which one is most similar to this input.

What does it mean for an input to be *most similar* to a neuron? The input pattern is a vector M consisting of k values. If we consider a neuron n , this neuron has k connections from the input neurons

m_i , which are assigned weights. The neuron n is thus represented by the vector of connection weights W_n .

We can then determine the Euclidean distance $dist_{Euclid}(M, n)$ between the pattern M and the neuron n :

$$dist_{Euclid}(M, n) = \sqrt{\sum_i (m_i - w_{in})^2} \quad \mathbf{6.14}$$

If these distances are now calculated for all neurons in the map layer, there is one neuron that has the smallest distance to the input pattern, see → Figure 6.29. This is the *winner neuron* z , also called *Best Matching Unit* (BMU), for which the following applies:

$$dist_{Euclid}(M, z) = \min_n dist_{Euclid}(M, n) \quad \mathbf{6.15}$$

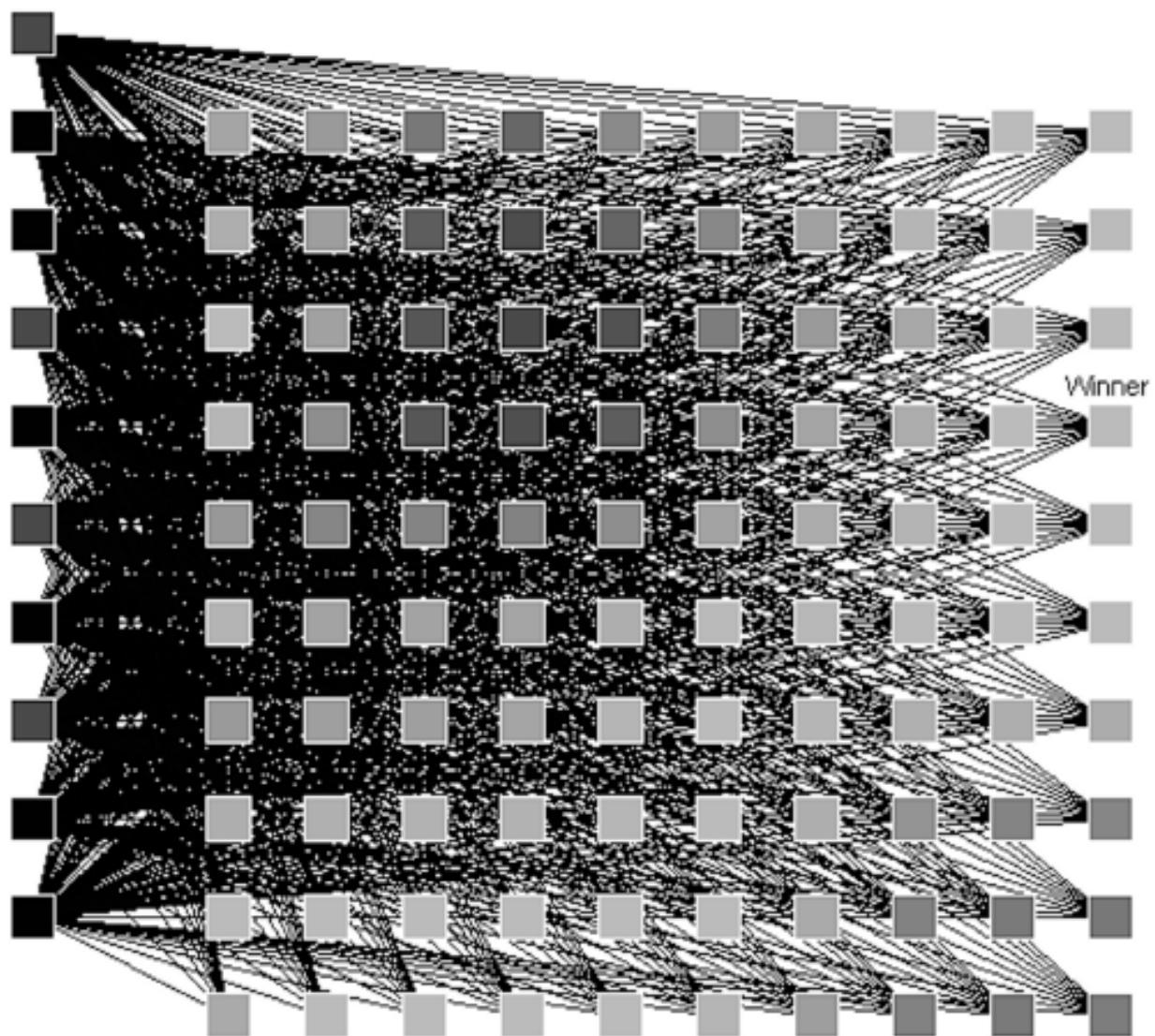


Fig. 6.29 Winner neuron in the map layer (2-dimensional grid)

Now the connection weights w_{iz} to the winner neuron are changed so that the winner neuron becomes even more similar to the input. In addition, connection weights from the input neurons to the neurons in the neighborhood of the winner neuron Z are also adjusted so that these vectors W_n also become more similar to the input M . The adjustment depends on the distance $dist(i, z)$ to the winner neuron z . Only neurons within a certain radius are changed. The weight change is usually calculated using the following formula:

$$w'_{ij} = \begin{cases} w_{ij} + \lambda \cdot h_{jz} \cdot (m_i - w_{ij}) & \text{if } \text{dist}(j, z) \leq r \\ w_{ij} & \text{otherwise} \end{cases} \quad 6.16$$

Here, λ is the learning parameter, and h influences the weight change depending on the distance of the neuron to the winner neuron. The neighborhood radius r determines the range within which the weights are changed. The weight change decreases with the distance of neuron i to winner neuron z :

$$h_{jz} = e^{-\frac{\text{dist}(j,z)^2}{2 \cdot r^2}} \quad 6.17$$

Clustering by means of a self-organizing map is performed according to the algorithm in → Listing 6.9. After each cycle, the learning parameter and the radius are reduced. At the beginning, the entire map is affected, and gradually only local changes are made. This ensures that the process terminates, as no further changes are made once the parameters have become very small (0).

```

PROCEDURE Kohonen learning
    Initialize all weights  $w_{ij}$ 
    REPEAT
        Select a pattern  $m$  at random
        Calculate the winner neuron  $z$  for pattern  $m$ 
        FOR all neurons  $j$  in the map layer DO
            IF  $\text{dist}(j, z) \leq r$  THEN
                
$$h_{ij} := e^{\frac{\text{dist}(j,z)^2}{2 \cdot r^2}}$$

                
$$w_{ij} := w_{ij} + \lambda \cdot h_{jz} \cdot (m_i - w_{ij})$$

            END IF
        END FOR
        Decrease learning factor  $\lambda$  and Radius  $r$ 
    UNTIL  $\lambda = 0$  OR  $r = 0$  OR number of iterations reached
END Kohonen learning

```

Listing 6.9 (Kohonen learning).

6.7.3 Visualization of a SOM

The principle of a self-organizing map can be illustrated very well, but preferably in a two-dimensional input space. All patterns then consist of exactly two values, which can be interpreted as x-y coordinates. The input pattern can thus be drawn as a point in the plane. Similarly, the connection weights from the two input neurons to a map neuron are interpreted as coordinates of this neuron. The neuron is represented as a point in the plane, see → Figure 6.30.

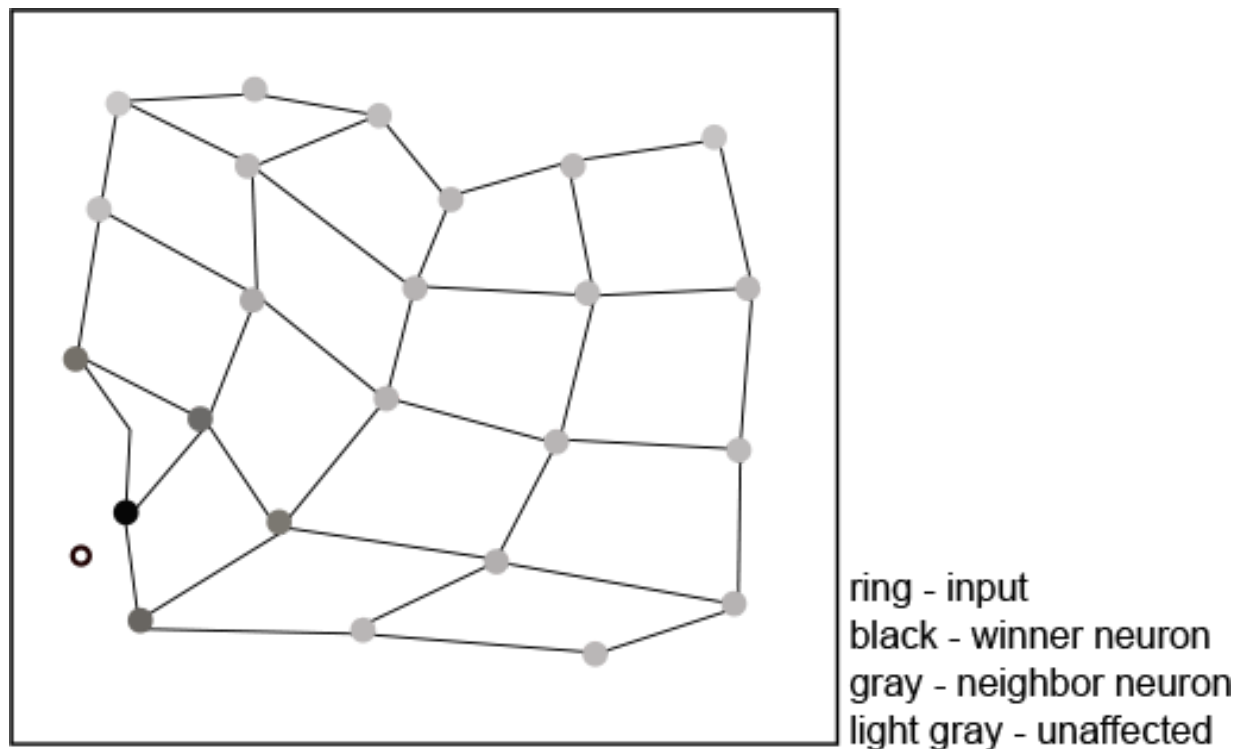


Fig. 6.30 Visualization of the map layer as points in the plane, the 2-dimensional space. The point outside is the current input, the black dot is the winner neuron, gray dots constitute the neighborhood, and the light gray dots are unaffected neurons.

The training of a self-organizing map is perceived as a change in the location of the neurons in the plane. At the beginning, the neurons are randomly distributed in the plane, because the connection weights are initialized with random values. During training, random points from the input space are presented to the network. This changes the connection weights, visually altering the position of the neuron. At the end of the learning process, the neurons of the map layer are evenly distributed across the input space. Each neuron then represents all points (inputs) within the corresponding Voronoi tile, see → Figure 6.31.

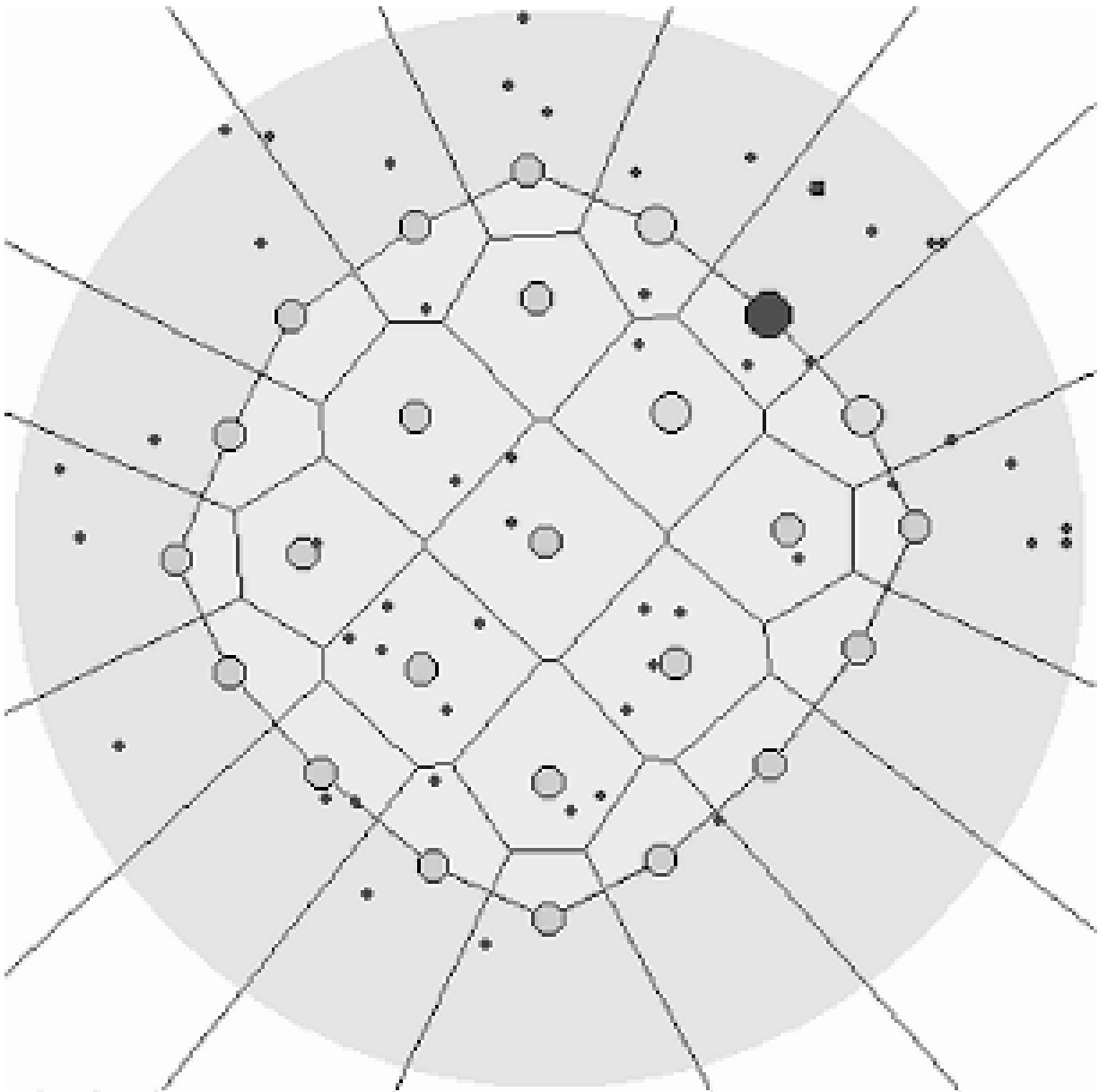


Fig. 6.31 Visualization of the map layer with Voronoi tiles
(DemoGNG → <https://www.demogng.de/>, March 25, 2026)

In the figure, the data can be seen as small dots, all of which originate from the circular input space that almost fills the image. The neurons are represented as small gray (green in the original) circles. The pattern currently being processed is a slightly larger square dot, and the corresponding winner neuron is located in the upper right area as a large dark (red in the original) circle. Each

neuron thus represents a cluster to which all patterns (dots) within the respective limited area, the Voronoi tile, are assigned.

6.7.4 An example

We refer to a illustrative, though some what older example, which according to Kruse [Kru+91] goes back to Ritter and Kohonen [→ RK89]. Animals are described based on their characteristics using binary vectors. We humans know the name so of the animals, but the self-organizing map does not. ASOM can only analyze the characteristics, which raises the question: Can a SOM recognize similarities between the animals and cluster them accordingly?

The characteristics and their meanings are listed in → Table 6.1, while → Table 6.2 contains the data records for the 13 animals.

Tab. 6.1 Characteristics of the animal data

Position	Feature	Position	Feature
1	small	8	mane
2	medium	9	feathers
3	big	10	hunts
4	2 legs	11	runs
5	4 legs	12	flies
6	hair	13	swims
7	hooves		

Tab. 6.2 Data for the animal example

Animal	1	2	3	4	5	6	7	8	9	10	11	12	13
Pigeon	1	0	0	1	0	0	0	0	1	0	0	1	0
Hen	1	0	0	1	0	0	0	0	1	0	0	0	0
Duck	1	0	0	1	0	0	0	0	1	0	0	0	1
Goose	1	0	0	1	0	0	0	0	1	0	0	1	1
Owl	1	0	0	1	0	0	0	0	1	1	0	1	0
Falcon	1	0	0	1	0	0	0	0	1	1	0	1	0
Eagle	0	1	0	1	0	0	0	0	1	1	0	1	0
Fox	0	1	0	0	1	1	0	0	0	1	0	0	0
Dog	0	1	0	0	1	1	0	0	0		1	0	0
Wolf	0	1	0	0	1	1	0	1	0	1	1	0	0
Cat	1	0	0	0	1	1	0	0	0	1	0	0	0
Tiger	0	0	1	0	1	1	0	0	0	1	1	0	0
Lion	0	0	1	0	1	1	0	1	0	1	1	0	0
Horse	0	0	1	0	1	1	1	1	0	0	1	0	0
Zebra	0	0	1	0	1	1	1	1	0	0	1	0	0
Cow	0	0	1	0	1	1	1	0	0	0	0	0	0

JAVANNS can be used to build a self-organizing map for the example. A corresponding input layer consisting of 13 neurons (type input) is generated for the input. The map itself can be created as a 10 x 10 neuron layer (type hidden). The Euclidean distance is selected as the activation function. Dark-colored neurons are closer to the input than light-colored neurons. This allows the excitation center to be visually identified. In the special Kohonen view in → Figure 6.32, the boundaries between the clusters are visualized.

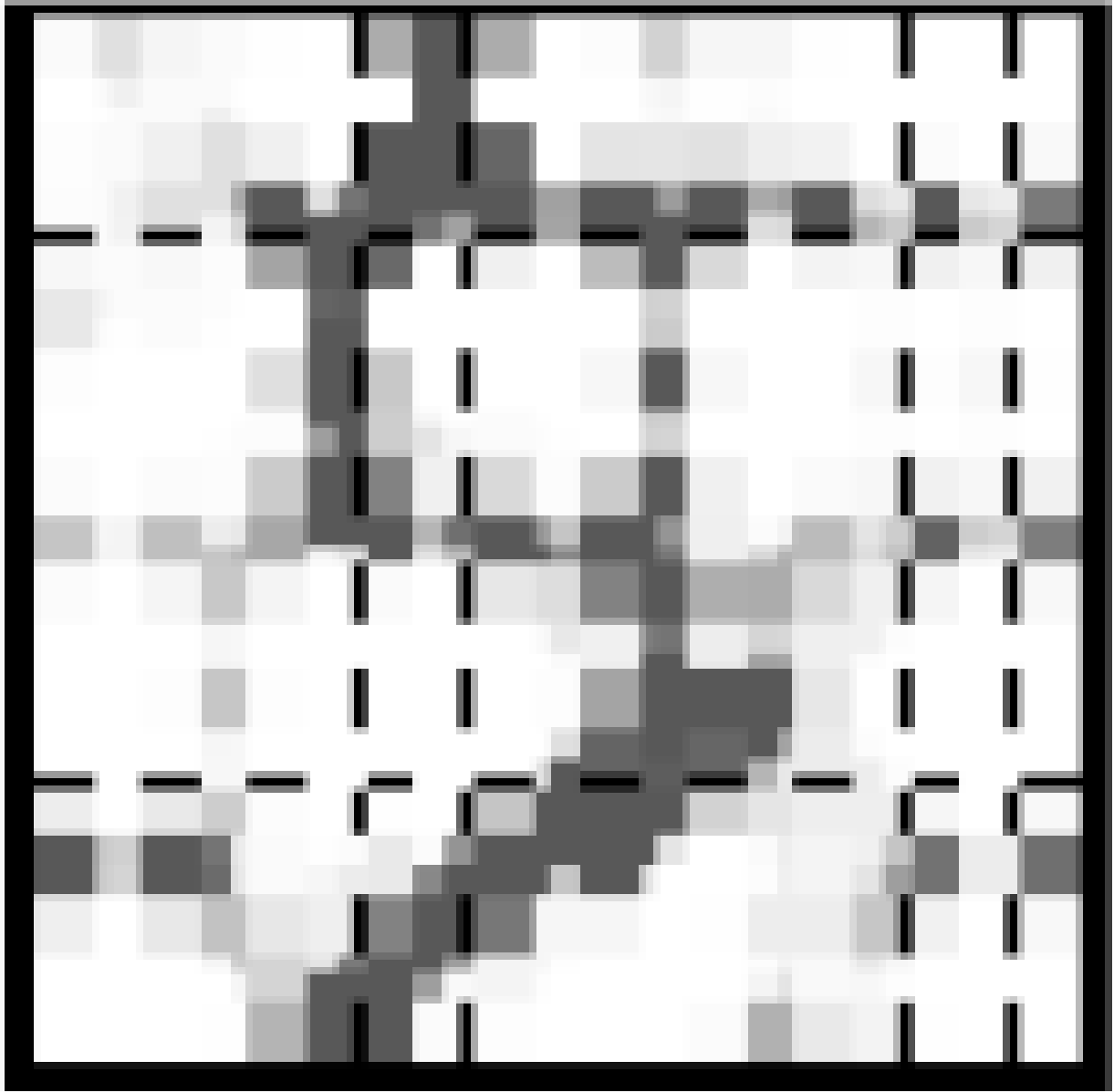


Fig. 6.32 Visualization of the clusters for the animal example in JAVANNS

A visualization that is more illustrative for smaller examples writes the name of the data record directly into the winner neuron, see → Figure 6.33.

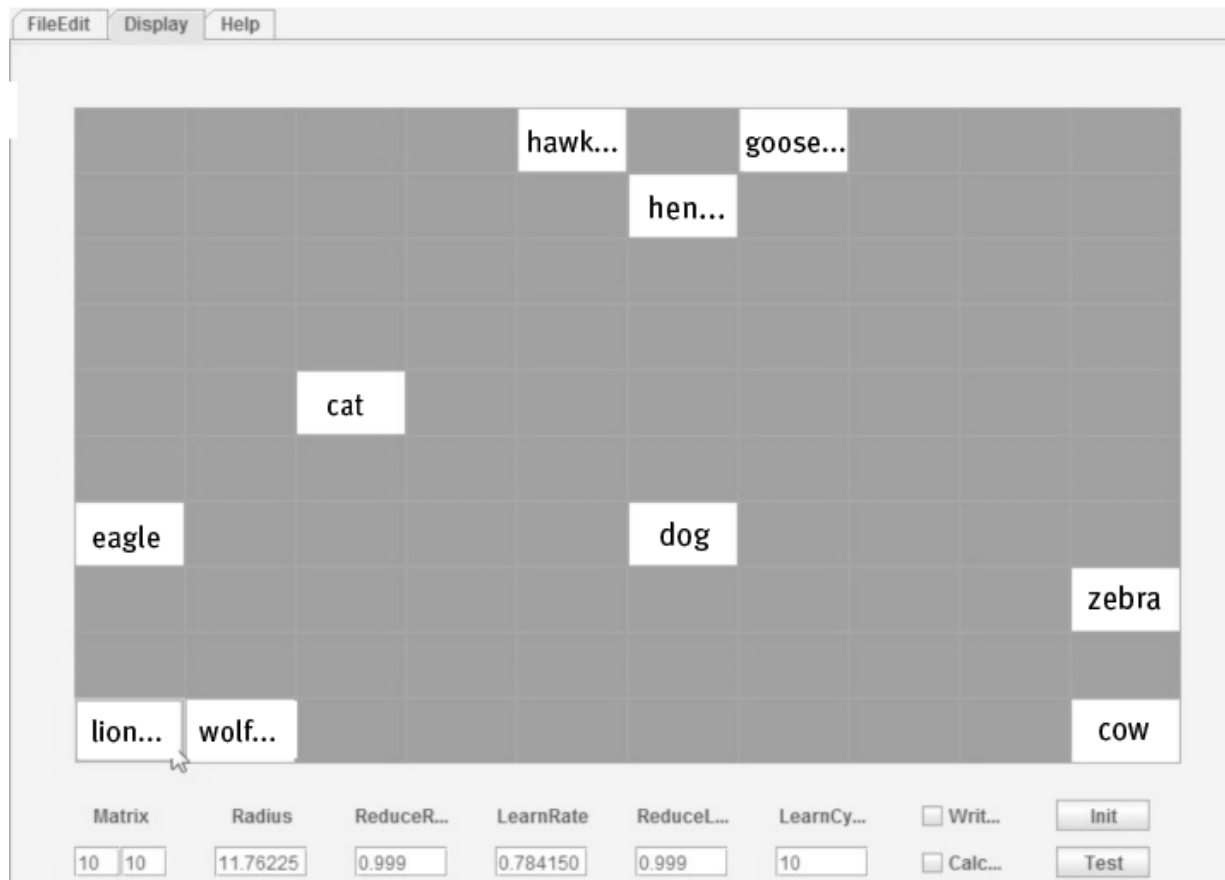


Fig. 6.33 Grouping of animals in a SOM

This shows that a self-organizing map can indeed recognize similarities on its own. Animals with similar characteristics are assigned to neighboring neurons. Clusters form on the map, which can be used after training to classify further data, in this example animals. *Horse* and *zebra* cannot be distinguished based on the characteristics considered in the example. They are mapped to the same winner neuron, and only one of the two names will be shown. The experiments were carried out using a small program developed at Wismar University.²

6.8 Clustering using neural gas

The term *neural gas* indicates that the concept is based on two models: on the one hand, it draws on the idea of artificial neural networks, and on the other hand, it regards the data records, points in multidimensional space, as particles in a gas. These particles have weak connections to other particles and can move around largely freely. From artificial neural networks, the concept of learning is transferred to neural gases.

A neural gas consists of a set of neurons that are not connected to each other. An input layer is also required to make an input, an input pattern or input vector, available to the neural gas. All neurons in the input layer are connected to each gas neuron n . Analogous to a self-organizing map, each gas neuron n has as many weighted input connections as the input has elements: k , see → Figure 6.34.

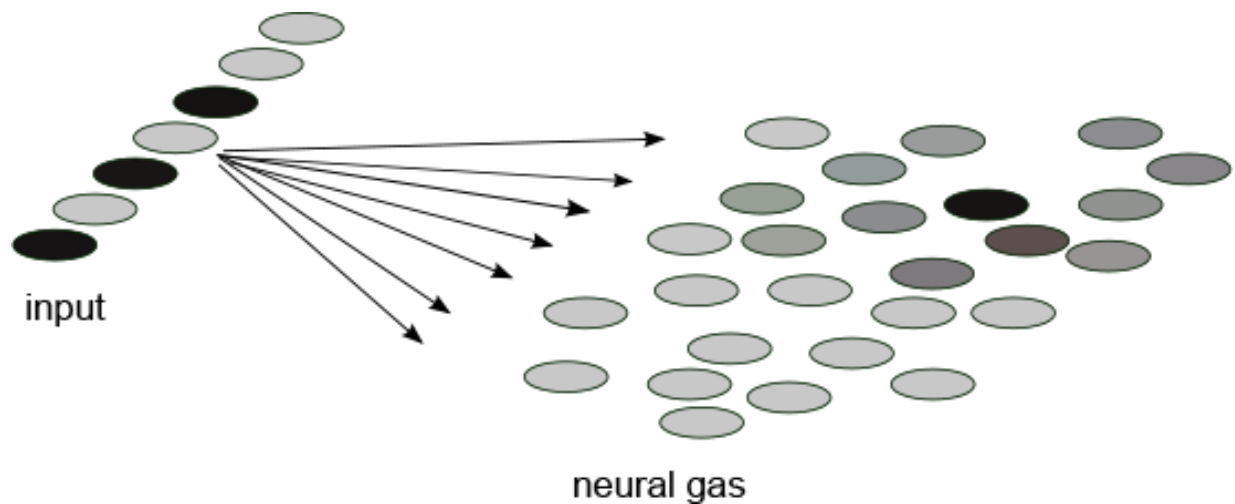


Fig. 6.34 Structure of a neural gas

A neural network is trained with input patterns for later use. In a neural network, so-called *soft competitive learning* is used. In *competitive learning*, a winner neuron is calculated for an input pattern, and the connection weights from the input to this winner

neuron are changed. The algorithm *Neural Gas Learning* (→ Listing 6.10) shows the procedure.

With this change, the input pattern X and the weight vector of the winner neuron n become somewhat more similar. By repeatedly applying all patterns, the winner neuron thus emerges as the center of a cluster. The specific formulas for changing the weights W_j as well as the learning factor and the neighborhood radius k can be found in [→ LC23] or [→ Fri98].

The following terms are used in the algorithm:

W_j : vector of connection weights from the input layer to the neuron j

X : input pattern (input vector), corresponds to the output of all neurons in the input layer.

PROCEDURE Neural Gas Learning

REPEAT

Randomly select pattern X , assign it to the input layer

Determine the winner neuron n for pattern X

Change weights W_n

Determine the k most similar neurons for pattern X

FOR $j := 1$ **TO** k **DO** Change weights w_j

Decrease learning parameters and k

UNTIL desired fit is achieved

END Neural Gas Learning

Listing 6.10 (Neural Gas Learning).

The *DemoGnG*³ demonstration software by Fritzke and Loos [→ Fri98] illustrates the learning process of a neural gas. At the same time, the two-dimensional case clearly shows how, at the end of the training process, the neurons emerge as the centers of clusters that cover the input space. → Figure 6.35 shows the result of the training process for a two-dimensional input space.

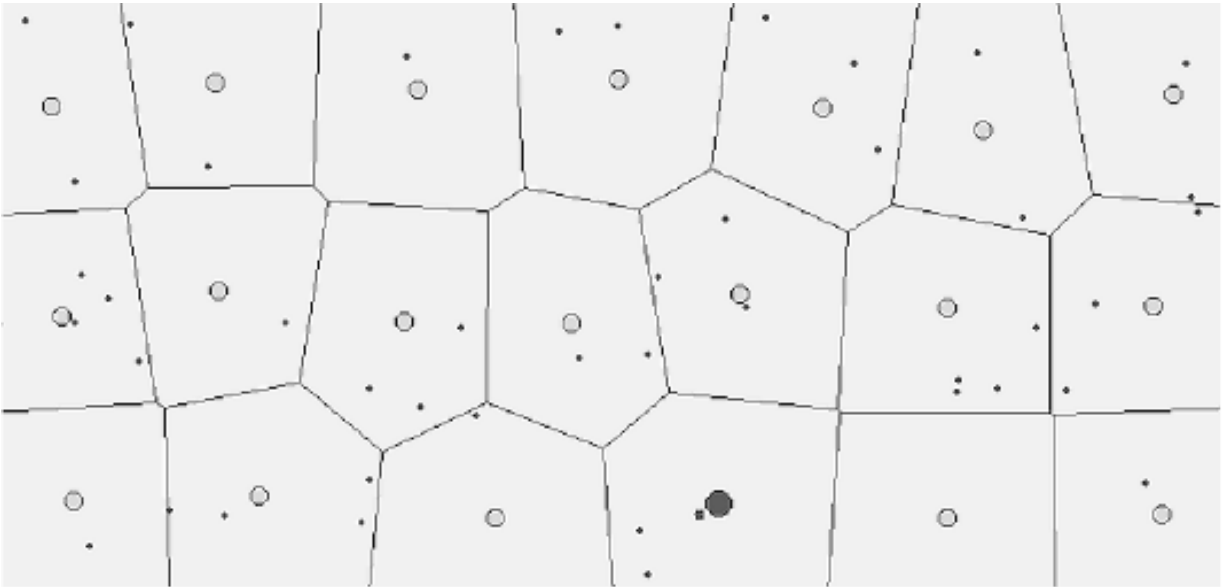


Fig. 6.35 The neurons of the neural gas cover the input space

A neural gas trained in this way can then be used to cluster data.

While a neural gas works with a fixed number of neurons, the number of neurons in a *growing neural gas* increases slowly. It starts with two neurons, and after a few training steps, a new neuron is inserted between the two neurons with the largest error. Again, this process can be easily understood in the DemoGNG software.

→ Figure 6.36 shows the growing neural gas at the beginning of the process on the left and in the state after the gas has already grown to 8 of a maximum of 10 neurons on the right. Details of the training algorithm can be found in [→ Fri98] or [→ Fri97].

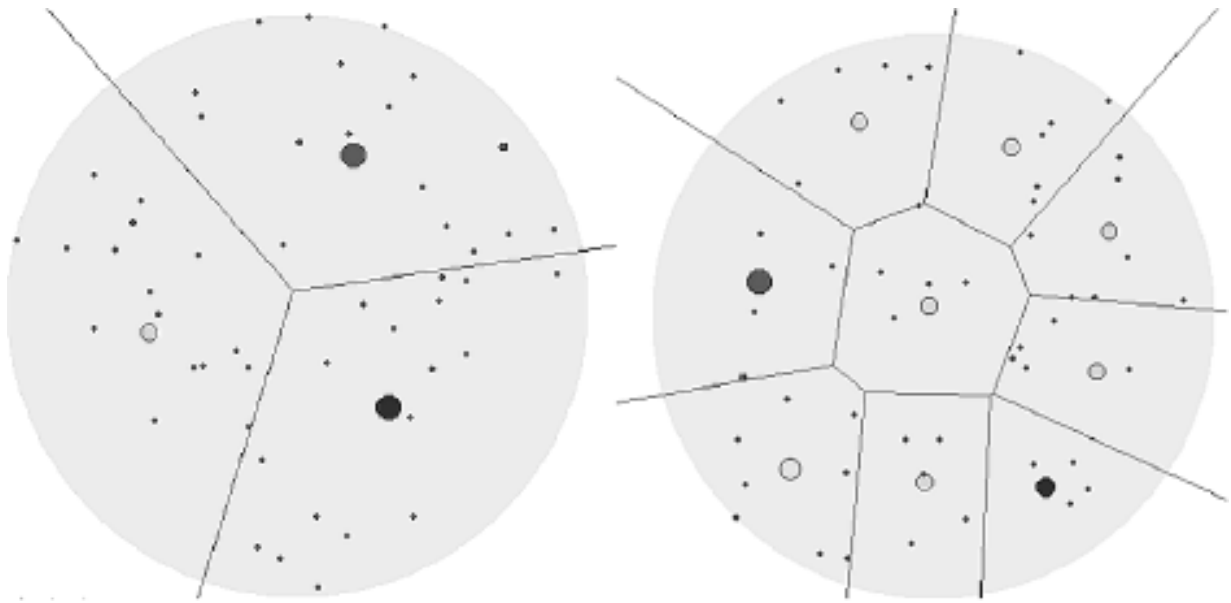


Fig. 6.36 Growing neural gas in two stages (DemoGNG)

6.9 Cluster formation using ART

Adaptive Resonance Theory (ART) is a particularly interesting approach to artificial neural networks that attempts to solve the so-called *plasticity-stability dilemma* of feedforward neural networks. The plasticity-stability dilemma consists in the fact that once a network has been trained, it cannot recognize patterns from other classes. For example, if a network has been trained to recognize all uppercase letters, it will not be able to classify lowercase letters, it doesn't have plasticity. If you train this network on lowercase letters afterwards, it loses the ability to recognize uppercase letters, it is not stable.

The idea behind the ART network is that it can recognize a certain set of patterns and at the same time is able to determine when a pattern cannot be assigned to any of the existing classes. In this case, a new class is created. This dynamic increase in the number of classes can be seen as a clustering process. An ART network can therefore do both. In an initial phase, it can be used for clustering: clusters are formed from the set of input patterns.

The same network can then also classify patterns. In doing so, it retains the ability to form a new cluster if necessary.

When is a new cluster formed? This question is answered by describing how an ART network works. The description is based on → Figure 6.37:

- An input is applied to the network.
- A winner neuron Z is determined for the input in the recognition layer F_2 .
- For each neuron already used in the recognition layer F_2 , there is a comparison pattern in the comparison layer F_1 . This pattern is the reference pattern for the class represented by Z .

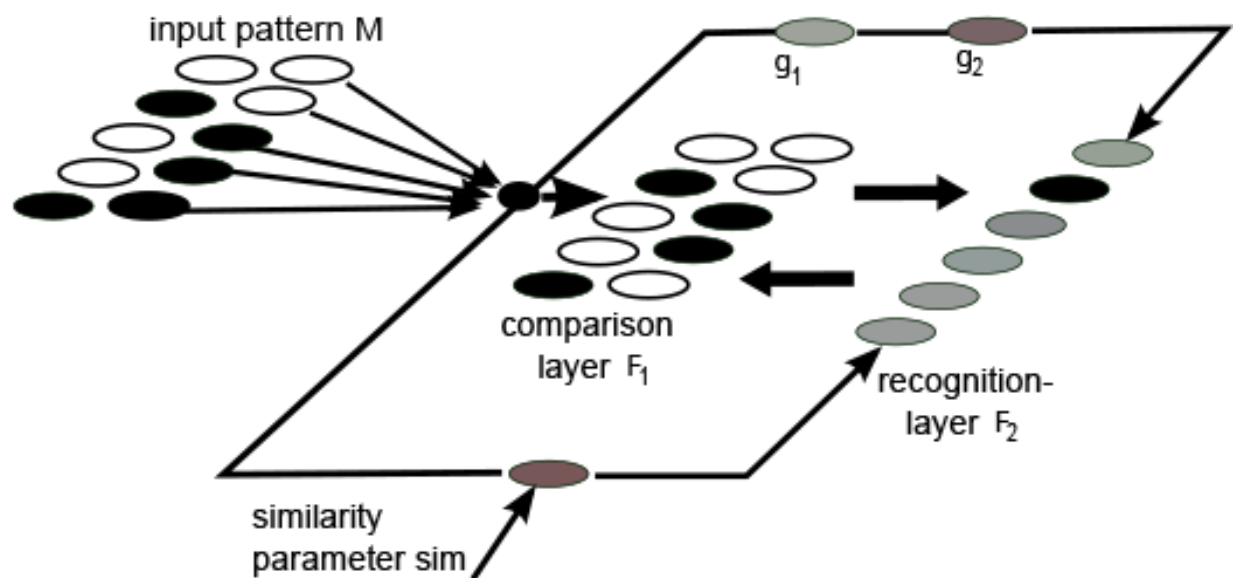


Fig. 6.37 Structure of an ART network

- Now, there are several possibilities:
1. The input is similar to the reference pattern. Then z is the class to which the input is to be assigned.

2. If the input is not similar to the reference pattern of z , the neuron is excluded from the competition, a new winner neuron is determined, and the comparison with the reference pattern is performed.
 3. If all neurons for which there is a reference pattern have been tried in this way and no similarity has been found, the current input does not belong to any of the existing classes. A new class is required. For this purpose, a neuron in the recognition layer that has not yet been used becomes the representative of the new class, and the input pattern is stored as the reference pattern for the new class.
- Once the pattern has been assigned to a class, the weights of the connections are modified and the reference pattern is changed so that it can also represent the current input pattern.

The concept of *similarity* is crucial to the functioning of an ART network. This similarity is specified by the user and passed to the network as the parameter *sim* ($0 < sim < 1$). A small value means that many patterns are considered similar to each other. A value close to 1 means that almost all different patterns are classified into different classes.

Among the examples in JAVANNS, there is also an ART network for recognizing uppercase letters, which are entered as a 5 x 7 pixel matrix. This example clearly illustrates the influence of the similarity parameter *sim* on classification and clustering. → Figure 6.38 shows the letter *M* as the input pattern and the corresponding reference pattern. With a similarity factor of $sim = 0.5$, all uppercase letters are divided into only 7 classes; if $sim = 0.8$ is set, 19 classes are distinguished. The user can thus control the number of clusters that are formed using the similarity parameter.

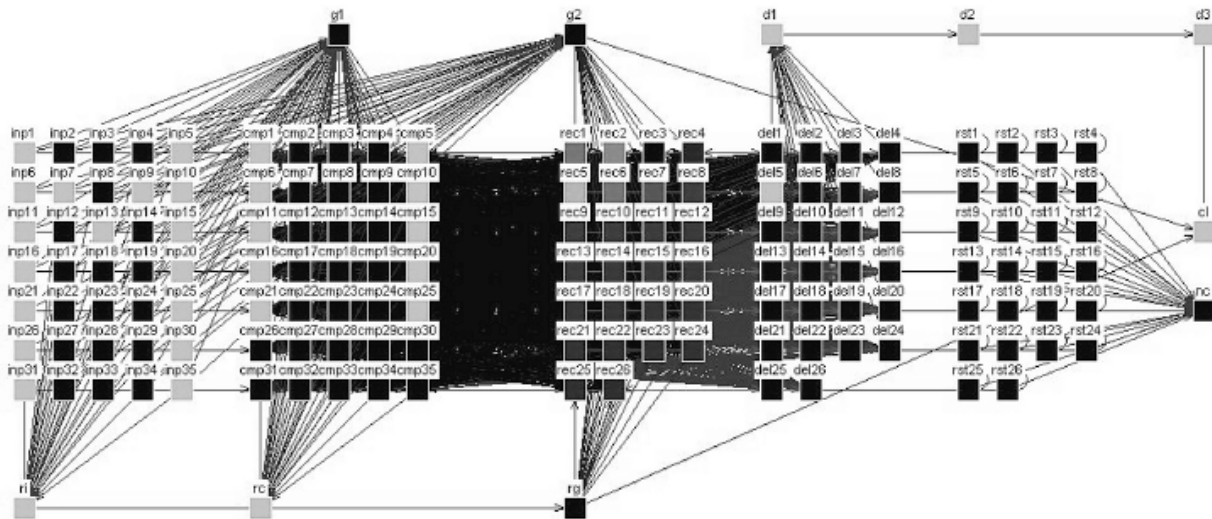


Fig. 6.38 Input pattern M and reference pattern in an ART network (JAVANNS)

ART networks are frequently discussed in the literature. Unfortunately, there is no easy-to-use software available that allows these networks to be used for realistic tasks.

6.10 The fuzzy c-means algorithm

To conclude the chapter on clusters, we return to the *k-means algorithm* (→ Section 6.2). The *k-means algorithm* has the disadvantage that it assigns each data record to exactly *one* cluster. However, we have already seen that this is not always useful. Let us consider the data shown in → Figure 6.39 in a two-dimensional space. We want to create two clusters.

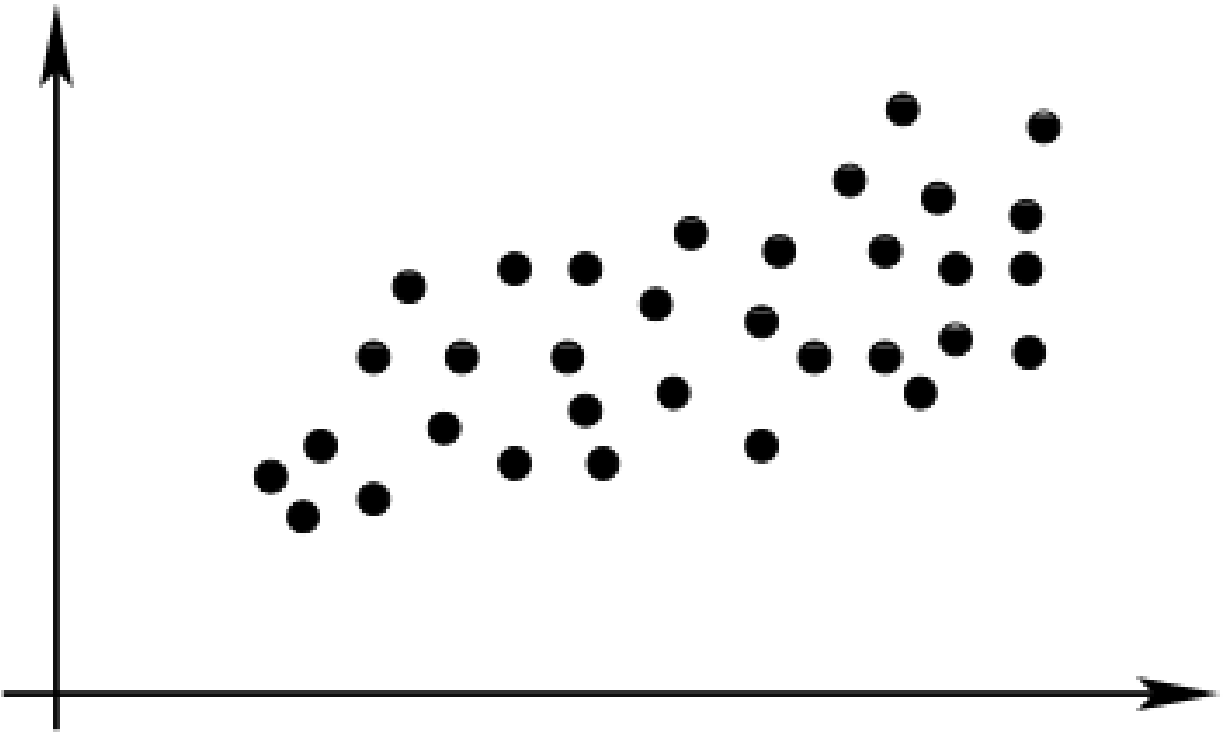


Fig. 6.39 Clustering in two dimensions using k-means

How can a meaningful boundary be drawn between the two clusters?

With density-based clustering (→ Section 6.6), we discussed an approach that can handle such point clouds. However, this does not help us here, as even the density-based clustering algorithm DBScan makes a hard decision as to whether a point belongs to a cluster or not.

In such a situation it is not appropriate to force the algorithm to make a decision. This is particularly questionable in the case of the *middle* points. If two clusters are to be formed from the given data, it makes sense to allow the points to belong to both clusters, but only partially.

This means that membership values between 0 and 1 are also allowed. This is a typical fuzzy approach. In doing so, we abandon the restriction of disjointness between the clusters (see → Section

3.1). We had already done this with the EM algorithm (→ Section 6.4).

The membership to the clusters can be represented using a matrix. We consider four-dimensional data ($\subset \mathbb{R}^4$) and assume that we have five data records that can be assigned to four clusters. The following matrix shows the membership of the five data records to the four clusters.

$$U = \begin{pmatrix} 0.1 & 0 & 0 & 0.9 \\ 0.2 & 0 & 0.8 & 0 \\ 0 & 1 & 0 & 0 \\ 0.8 & 0 & 0.2 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

The rows show the membership of the respective data record to the respective cluster. u_{ij} is the affiliation of the data record x_i to the cluster C_j .

Without listing the data here, we can see which clusters they belong to. According to the matrix above, the first data record belongs 10 % to the first cluster C_1 and 90 % to cluster C_4 . Similarly, the fourth data record is assigned 80 % to cluster C_1 and 20 % to cluster C_3 . The example shows that a clear assignment is definitely possible (data records 3 and 5). Cluster C_2 is a cluster that would also have been formed by the k-means algorithm.

If k is the number of clusters to be formed and n is the number of data points to be clustered, then the matrix U has the size $n \times k$. The element u_{ij} then describes the membership of the data record x_i to the cluster C_j .

In addition we require that the sum of all membership values of a data record x_i to the clusters is equal to 1 :

$$\sum_{j=1}^k u_{ij} = 1 \quad \text{for all } i$$

Empty clusters are not permitted:

$$\sum_{i=1}^n u_{ji} > 0 \quad \text{for all } j$$

Each cluster C_j thus has at least one element, albeit possibly with only a low degree of membership.

The goal of fuzzy c-means is to form the best possible clusters from the given data. The following sum serves as the quality function (see → Section 9.5):

$$Q = \sum_{i=1}^n \sum_{j=1}^k u_{ij}^m \cdot \text{dist}(x_i, c_j)^2 \quad 6.18$$

The weights u_{ij} are the values contained in the matrix U . Here, $\text{dist}(x_i, C_j)$ is the Euclidean distance between x_i and the cluster center c_j of cluster C_j . The value of m controls how soft or hard the clustering is. m must be greater than 1.

Now let's turn to the actual algorithm, the fuzzy c-means method, which was developed by Dunn [→ Dun73] and Bezdek [→ Bez81].

How can we find the optimal membership values u_{ij} ? Obviously, this is an optimization problem. We want to minimize the quality function (6.18), while additionally fulfilling the constraints. We apply the Lagrange method here and obtain, omitting the mathematical details, the cluster centers

$$c_j = \frac{\sum_{i=1}^n u_{ij}^m \cdot x_i}{\sum_{i=1}^n u_{ij}^m} \quad 6.19$$

and the new membership values:

$$u_{ij} = \frac{1}{\sum_{s=1}^k \left(\frac{\text{dist}(x_i, c_j)^{\frac{2}{m-1}}}{\text{dist}(x_i, c_s)^{\frac{2}{m-1}}} \right)} \quad 6.20$$

The parameter m determines the sharpness of the clustering. If m is allowed to run to infinity, the u_{ij} converge to $\frac{1}{k}$. If, on the other hand, m approaches 1, sharp clusters are obtained. The membership values are then close to 0 or 1. A value of $m = 2$ has proven to be a favorable starting value for experiments.

1. Initialize U .
2. Calculate the cluster centers C_j according to Equation 6.19.
3. Calculate the new membership matrix U according to Equation 6.20.
4. If the improvement of U is small ($< \epsilon$), then stop, otherwise go to 2.

Listing 6.11 (Fuzzy-c means).

The initialization of U can be done randomly, for example. The parameter ϵ controls the end of the process.

Finally, if we look at the Iris data (see → Appendix A.1) and apply the fuzzy c-means algorithm available in KNIME to the normalized data, we get the output shown in → Figure 6.40.

Row ID	D sepal...	D sepalwi...	D petal...	D petalw...	S class	D cluster_0	D cluster_1	D cluster_2	S Winner ...
48	0.278	0.708	0.085	0.042	Iris-setosa				cluster_2
49	0.194	0.542	0.068	0.042	Iris-setosa				cluster_2
50	0.75	0.5	0.627	0.542	Iris-versic...				cluster_0
51	0.583	0.5	0.593	0.583	Iris-versic...				cluster_1
52	0.722	0.458	0.661	0.583	Iris-versic...				cluster_0
53	0.333	0.125	0.508	0.5	Iris-versic...				cluster_1
54	0.611	0.333	0.61	0.583	Iris-versic...				cluster_1
55	0.389	0.333	0.593	0.5	Iris-versic...				cluster_1
56	0.556	0.542	0.627	0.625	Iris-versic...				cluster_1
57	0.167	0.167	0.39	0.375	Iris-versic...				cluster_1
58	0.639	0.375	0.61	0.5	Iris-versic...				cluster_1
59	0.25	0.292	0.492	0.542	Iris-versic...				cluster_1

Fig. 6.40 Clustering of the Iris data using fuzzy c-means

The membership values are displayed graphically here. KNIME also provides the winning cluster, i.e., the cluster with the highest membership value.

7 Association analysis

The one who says it cannot be done should never interrupt the one who is doing it.

The Roman Rule

The aim of *association analysis* (association rule mining) is to find association rules (see → Sections 3.4 and → 4.4). Association rules are similar to classification rules, except that their predictions are not limited to the target attribute, but rather establish relationships between arbitrary attributes. Association analysis is a form of *unsupervised* learning (→ Section 2.6). Relationships between attributes are sought on the basis of the given examples.

In a sense, association analysis can also be classified as *supervised* learning, since a model (a set of association rules) is developed based on past data, which can be verified using further data (test set or new data). The search for association rules has led to the development of a separate subfield: *Association rule mining* (ARM).

One of the main areas of application for association analysis is *market basket analysis* (also known as affinity analysis). An association rule of the form “Those who buy A also buy B” is extremely valuable for a retail company. The goods in a supermarket can then be arranged so that A and B are not placed too far apart and products that are also likely to be purchased together with A are placed between them.

The standard method for association analysis is the *Apriori algorithm*, for which there are a number of variants. Another method we consider is *FP growth*, which also searches for the frequent co-occurrences of items.

7.1 The Apriori algorithm

The Apriori algorithm [→ AS94] is one of the central iterative methods for generating *association rules*. It represents a further development of the AIS algorithm, which was published in 1993 [→ AIS93]. The aim of the Apriori algorithm is to find *frequent item sets* in the total set of items. Any object can be referred to as an *item*; in a shopping cart, these are the articles or products.

For the objects in a *frequent item set*, their frequency (their support, see → Sections 4.4 and → 9.2.1) exceeds a certain threshold value specified by the analyst. If the minimum support is set at 2 %, the set {beer, bread, butter} is a frequent item set if this combination of three items occurs in at least 2 % of all purchases.

Why is it sufficient to limit the search for association rules to *frequent item sets*? The rule

Those who buy A and B also buy C.

fulfills the required minimum support precisely when the set {A, B, C} is a frequent item set. This means that support does not depend on the structure of the rule, but only on the items it contains. The rule

Those who buy C and A also buy B.

therefore has the same support. If the set {A, B, C} is *not* a frequent item set, no rule containing A, B, and C can fulfill the support requirement. It therefore makes no sense to immediately search for rules that meet the minimum support requirement. Instead, one should first search for frequent item sets.

The Apriori algorithm is executed in two steps:

1. Finding *frequent item sets* with sufficient support:
First, all sets of products that meet the required support are searched for. In market basket analysis, these are all product

sets that have been purchased together often enough according to the support.

2. Generating *association rules* from all frequent item sets:
In the second step, *rules* are generated from these frequent item sets.

When generating association rules in step 2, the Apriori algorithm focuses exclusively on the use of previously found frequent item sets. This limits the search space and thus reduces the complexity of the process.

Let us consider step 1, finding frequent item sets. We start with *single-element* frequent item sets. In the next pass, the single-element frequent item sets found are used to generate *two-element* frequent item sets, and in the following pass, the two-element item sets are used to generate *three-element* item sets. The process is repeated until no more frequent item sets are found. The frequent item sets found are referred to as *candidates*.

The support is calculated for each candidate. If it is less than the threshold value, the candidate is rejected.

To further reduce the number of candidates and improve runtime performance, the *monotonicity property*

$$M \subset N \rightarrow \text{supp}(M) \geq \text{supp}(N) \quad 7.1$$

of the item sets is used. It states that every non-empty subset M of a frequent item set N must itself be a frequent item set. This means, that the support cannot become *smaller* if an item is removed from a candidate frequent item set. Conversely, if an item is added to a frequent item set, the support *cannot increase*.

Thus, two-element frequent item sets can only consist of those single-element frequent item sets that meet the required support. According to monotonicity, the support does not increase when another item is added to an item set. Therefore, if a single-element frequent item set does not meet the support requirement, a multi-

element item set containing this item cannot meet the required support.

Similarly, when generating sets of four candidates, we do not need to examine every possible combination of four items, but only those sets of four that are derived from two sets of three candidates. If we have the frequent item sets $\{A, B, C\}$ and $\{A, C, D\}$, then $\{A, B, C, D\}$ can be a frequent item set. We must consider this candidate. However, if we know that $\{B, C, D\}$ was not a three-element frequent item set, then $\{A, B, C, D\}$ cannot be a frequent item set due to monotonicity. This means we don't have to count in our shopping cart database whether our candidate fulfills the support requirement. We know in advance that the support will not be fulfilled. This saves a lot of computing time.

7.1.1 Generating the candidates

Let's now take a closer look at how candidates are generated. This is done in two phases:

1. Join phase
2. Pruning phase

In the join phase, all $(k - 1)$ -element item sets that differ in only one element are joined in pairs. This results in k -element item sets, for which it is ensured that at least two subsets of size $k - 1$ are also frequent item sets.

In the pruning phase, the k -element candidates are checked to see whether *each* $(k - 1)$ -element subset was also a frequent item set. If this is not the case, i.e., if even one subset is not a frequent item set, then the k -element candidate cannot be a frequent item set due to the monotonicity property. The remaining candidates must then be checked to see whether they have the required minimum support.

The algorithm is shown in → Listing 7.1. We will abbreviate the term *frequent item set* as *FIS*. For the set of n -element frequent item

sets, we use the abbreviation FIS_n .

```

PROCEDURE GenerateFIS(minsupp, SetOfItemsets)
 $FIS_1 := \{\{x\} \mid \text{supp}(\{x\}) \geq \text{minsupp}\}$  // Calculate all
 $FIS$  of size 1
     $k := 1$ 
    WHILE  $FIS_k \neq \emptyset$ 
         $k := k + 1$ 
        Calculate  $FIS_k := \{a \cup b \mid a, b \in FIS_{k-1}, |a \cup b| =$ 
             $k\}$  // Join-Phase
        FORALL  $c \in FIS_k$  // Pruning: Are all  $(k-1)$ -
            subsets in  $FIS_{k-1}$ ?
            IF  $x \subset c$  exists with  $|x| = k - 1$  and  $x \notin FIS_{k-1}$ 
                THEN delete  $c$  in  $FIS_k$ 
        FORALL  $c \in FIS_k$  // Is the support fulfilled?
            IF  $\text{supp}\{c\} < \text{minsupp}$  THEN delete  $c$  in  $FIS_k$ 
        RETURN  $(\cup_i FIS_i)$ 
    END GenerateFIS

```

Listing 7.1 (Apriori, generation of frequent item sets FIS).

The support test for a candidate is performed immediately after the pruning phase. For reasons of clarity, these steps are separated in the algorithm.

When calculating subsets, it makes sense to temporarily store partial results such as support and confidence in order to avoid frequent access to the database.

Example 7.1

(Apriori, Movie theater visits).

We look at movie theater visits and examine who likes to go to the cinema with whom.

We require a minimum support of 50 %.

Movie theater visit ID	Visitors
k1	Anne, Clara, Ellen
k2	Anne, Ellen, Gerald
k3	Anne, Clara, Ellen, Frank, Gerald
k4	Anne, Clara, Harry
k5	Ben, Clara, Ellen, Frank, Gerald
k6	Ben, Clara, Ellen, Gerald, Harry

Visitor	Count	Support
Anne	4	66%
Ben	2	33%
Clara	5	83%
Ellen	5	83%
Frank	2	33%
Gerald	4	66 %
Harry	2	33 %

We start by counting the support for each individual. To do this, we count the relative frequency of the individuals in our table with the six data records.

What do we see? Ben, Frank, and Harry do *not* meet the minimum support requirement. When forming two-element frequent item sets, we can ignore these three visitors because every frequent item set of size 2 with one of these three individuals is guaranteed to have a support of at most 33 %.

Again, we denote the set of frequent item sets of size n by FIS_n and first form the candidates for FIS_2 .

Candidates for FIS_2	Support	No.
{Anne, Clara}	50%	1
{Anne, Ellen}	50%	2
{Anne, Gerald}	33 %	3
{Clara, Ellen}	66 %	4
{Clara, Gerald}	50 %	5
{Ellen, Gerald}	66 %	6

When moving from FIS_1 to FIS_2 , no pruning is necessary, as the proper subsets of all candidates from FIS_2 correspond exactly to the elements from FIS_1 . We calculate the support for all candidates. One of these six candidates, {Anne, Gerald}, does not meet the support requirement.

Now we form candidates for $FFIS_3$ by combining frequent item sets from FIS_2 in such a way that a triple candidate is created. What combinations can we form?

Candidates for FIS_3	Combination	Support
{Anne, Clara, Ellen}	1+2, 1+4, 2+4	?%
{Anne, Clara, Gerald}	1+5	?%
{Anne, Ellen, Gerald}	2+6	?%
{Clara, Ellen, Gerald}	4+5, 4+6, 5+6	?%

Do we now have to count again to determine the support? *No*, we can immediately exclude the two middle candidates, since {Anne, Gerald} is *not* a two-element frequent item set and therefore the candidates {Anne, Clara, Gerald} and {Anne, Ellen, Gerald} cannot have the required support due to the monotonicity of the support.

Our candidate {Anne, Clara, Ellen} (here we have to count) does not meet the minimum support of 50 %, whereas {Clara, Ellen, Gerald} meets the required support with 50 %. There cannot be any four-item candidates, since $|FIS_3| = 1$; so we only have one three-

element FIS. With the required support, we can determine five two-element frequent item sets and one three-element frequent item set from the sample data.

7.1.2 Generating the rules

After generating the frequent item sets, we split each frequent item set Y into two non-empty subsets X and $Y \setminus X$ and consider the rule

$$X \rightarrow (Y \setminus X)$$

From $Y = \{a, b, c, d\}$ and with $X = \{a, b\}$, the rule $(a, b) \rightarrow (c, d)$ can be formed. We now have to check whether this rule satisfies the *minimum confidence*. Confidence (see → Sections 9.2.2 and → 4.4) is defined as

$$\text{conf}\left(X \rightarrow (Y \setminus X)\right) = \frac{\text{supp}(Y)}{\text{supp}(X)} \quad 7.2$$

Confidence is defined in terms of support, and since we recorded the support for each frequent item set in the previous step, it is not necessary to count again in the database.

Of course, all rules resulting from the possible decompositions of Y must be checked. For each frequent item set Y , association rules of the form $X \rightarrow (Y \setminus X)$ are sought with $X \subset Y$ and $X \neq \emptyset$ that satisfy the confidence condition:

$$\text{conf}\left(X \rightarrow (Y \setminus X)\right) = \frac{\text{supp}(Y)}{\text{supp}(X)} \geq \text{conf}_{\min} \quad 7.3$$

It is not necessary to check all possible association rules for their confidence. Let us consider rules that contain these items: $Y = \{a, b, c, d\}$. The support of

$$X = \{a, b, c\}$$

is always less than or equal to the support of the item set

$$X' = \{a, b\}$$

Thus, the confidence of the rule

$$X' \rightarrow (Y \setminus X')$$

cannot be higher than the confidence of the rule

$$X \rightarrow (Y \setminus X)$$

Replacing X with X' does not increase confidence. If the rule $X \rightarrow (Y \setminus X)$ does not meet the minimum confidence, it is not necessary to consider rules of the form $X' \rightarrow (Y \setminus X')$ with $X' \subset X$.

The result contains all association rules that meet both the minimum support and the minimum confidence.

What are the *advantages* of this method? It involves simple set operations, which means implementation is not complicated. The restrictive approach in the join phase drastically reduces complexity. Consider how many rule candidates arise when simply trying out all possible combinations (for example, with 1 million sales products).

The *disadvantage* is that although complexity is drastically reduced, runtime problems still arise with large amounts of data due to frequent support calculations, which generally require accessing a database.

Example 7.2

(Apriori, Movie theater visit). Let's continue with our example above and consider the only triple candidate for illustration purposes: {Clara, Ellen, Gerald}. We pick only one rule to calculate the confidence:

$$\text{Clara, Ellen} \rightarrow \text{Gerald}$$

We have already calculated the support as 50 %. How high is the confidence? We apply the definition of confidence

$$\text{conf}(M \rightarrow N) = P(N|M) = \frac{\text{supp}(M \cup N)}{\text{supp}(M)}$$

and obtain

$$\text{conf}(\text{Clara, Ellen} \rightarrow \text{Gerald}) = \frac{50}{66} = 0.75$$

Since we have already calculated the support values, we do not need to access our database again to obtain the confidence.

How many times did we have to count in our database to determine the support? During the *FIS* generation phase (see → Example 7.1), we accessed our database:

- Determination of single-element frequent item sets *FIS*₁: 7
- Determination of two-element frequent item sets *FIS*₂: 6
- Determination of three-element frequent item sets *FIS*₃: 2
- Determination of four-element frequent item sets *FIS*₄: 0

With a naive approach, we would have had to try a total of 21 variants just to find frequent item sets of size 2, and as many as 35 for *FIS*₃ and *FIS*₄ respectively. The number of possible rules, even if only one item is placed on the right-hand side of the rule, increases the complexity dramatically. Since the complexity of this procedure is determined solely by the frequent accesses to the database in which the item sets are stored, the savings are obvious.

Example 7.3

(Frequent Item set). Suppose the following frequent item sets of size 3 have already been calculated. We therefore skip the generation of the sets FIS_1 , FIS_2 and FIS_3 and assume the following frequent item sets of size 3 as given:

$\{a, b, c\}$ (1)	$\{a, c, e\}$ (4)
$\{a, b, d\}$ (2)	$\{b, c, d\}$ (5)
$\{a, c, d\}$ (3)	

This generates the following *four-element candidates (join)*:

$\{a, b, c, d\}$ (1+2)
$\{a, c, d, e\}$ (3+4)
$\{a, b, c, e\}$ (1+4)

Pruning eliminates $\{a, c, d, e\}$ and $\{a, b, c, e\}$, as these sets each contain one $(k-1)$ subset ($\{c, d, e\}$ and $\{b, c, e\}$, respectively) that are not three-element frequent item sets.

So far, we have only applied association rules to Boolean data: our data records may only contain attributes that are true or false: butter is bought, Clara goes to the cinema. Can association rules also be found for data records of the following type?

Age	Pulse	Blood pressure	Condition
<40	irregular	normal	healthy
> 40	irregular	normal	ill
< 40	regular	normal	healthy
> 40	irregular	abnormal	ill
< 40	irregular	normal	ill
< 40	regular	normal	healthy
> 40	regular	abnormal	ill
< 40	regular	normal	healthy
> 40	regular	normal	healthy

At first glance, the Apriori algorithm cannot be applied to this data. However, if we convert the attributes into *binary* attributes (see Section 8.2.5), the algorithm can be used without any problems.

Age <40	Age ≥ 40	Pulse irregular	Pulse regular	Blood pressure abnormal	Blood pressure normal	Condition ill	Condition healthy
x		x			x		x
	x	x			x	x	
x			x		x		x
	x	x		x		x	
x		x			x	x	
x			x		x		x
	x		x	x		x	
x			x		x		x
	x		x		x		x

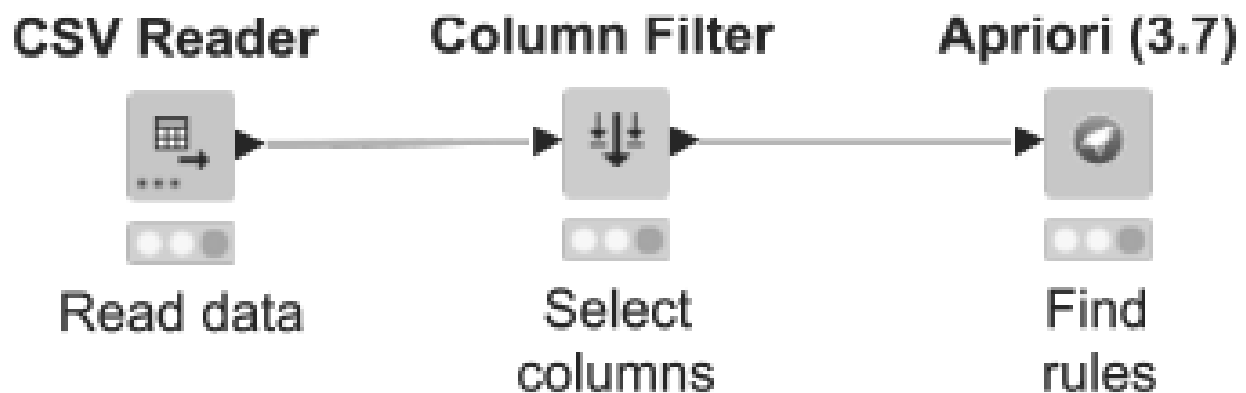


Fig. 7.1 Apriori workflow, variant 1

KNIME finds the following association rules in these data records using the workflow shown in → Figure 7.1.

Apriori

=====

Minimum support: 0.6 (5 instances)

Minimum metric <confidence>: 0.9

Number of cycles performed: 8

Generated sets of large itemsets:

Size of set of large itemsets L(1): 8

Size of set of large itemsets L(2): 12

Size of set of large itemsets L(3): 8

Size of set of large itemsets L(4): 2

Best rules found:

1. normal=y 7 ==> abnormal=n 7 conf:(1)
 2. abnormal=n 7 ==> normal=y 7 conf:(1)
 3. ge40=n 5 ==> less40=y 5 conf:(1)
 4. less40=y 5 ==> ge40=n 5 conf:(1)
 5. less40=y 5 ==> abnormal=n 5 conf:(1)
 6. less40=y 5 ==> normal=y 5 conf:(1)
 7. ge40=n 5 ==> abnormal=n 5 conf:(1)
 8. ge40=n 5 ==> normal=y 5 conf:(1)
-

It is noticeable that binary coding generates a number of nonsensical rules such as the rules 1 to 4. Although this can be corrected retrospectively using appropriate filters, it is unnecessary. Since we already have exclusively binary attributes, it is sufficient to represent our data as follows:

Age ≥40	Pulse regular	Blood pressure normal	Condition healthy
n	n	y	y
y	n	y	n
n	y	y	y
y	n	n	n
n	n	y	n
n	y	y	y
y	y	n	n
n	y	y	y
y	y	y	y

Now KNIME finds a set of useful rules that have a minimum support of 30 % and a minimum confidence of 90 %:

```

1.ge40=n 5 ==> normal=y 5
conf:(1)
2.healthy=y 5 ==> normal=y 5
conf:(1)
3.ge40=n healthy=y 4 ==> normal=y 4
conf:(1)
4.regular=y healthy=y 4 ==> normal=y 4
conf:(1)
5.regular=y normal=y 4 ==> healthy=y 4
conf:(1)
6.ge40=n regular=y 3 ==> normal=y 3
conf:(1)
7.ge40=n regular=y 3 ==> healthy=y 3
conf:(1)
8.ge40=n regular=y healthy=y 3 ==> normal=y 3
conf:(1)
9.ge40=n regular=y normal=y 3 ==> healthy=y 3
conf:(1)

```



```
10.ge40=n regular=y 3 ==> normal=y healthy=y 3
conf:(1)
```

The procedure is not limited to binary attributes. If, in addition to other attributes, there is an attribute *age* with the values *young*, *middle*, and *old*, the Apriori algorithm begins with the item sets *age=young*, *age=middle*, and *age=old*. To calculate the support and confidence, the number of times the attribute has the corresponding value is counted.

KNIME can also work with the original data from the table (Section 7.1, p. 187). To do this, the item sets are formed using *Create Collection Column*, see → Figure 7.2.

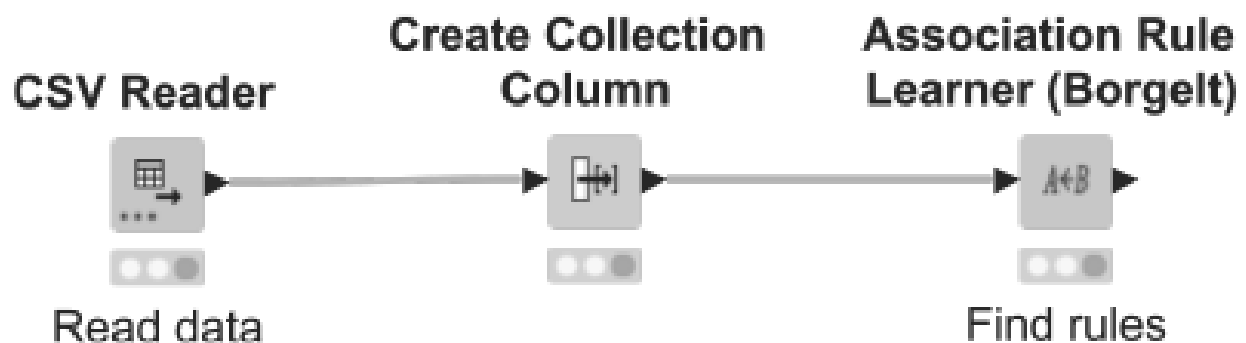


Fig. 7.2 Apriori workflow, variant 2

Some association rules can also be interpreted as classification rules (see → Sections 3.2 and → 4.3). If we want to predict the state (healthy or ill), we limit the right side of the if-then rules to the attribute *condition*. Of the above rules, the following are then of interest:

```
5. regular=y normal=y 4 ==> healthy=y 4
conf:(1)
7. ge40=n regular=y 3 ==> healthy=y 3
conf:(1)
```

9. ge40=n regular=y normal=y 3 ==> healthy=y 3

conf:(1)

10. ge40=n regular=y 3 ==> normal=y healthy=y 3

conf:(1)

However, these rules do not allow us to predict the patient's condition in every case. In this respect, association rules rarely enable us to achieve a complete classification. However, classification is possible for subsets at least.

Task 7.1 (Association analysis, Apriori).

Given the purchases listed in the following table:

ID	Purchased items
t ₁	bread, juice, cola, beer
t ₂	juice, cola, wine
t ₃	bread, juice, water
t ₄	cola, beer, juice
t ₅	bread, juice, cola, beer, wine
t ₆	water

First, complete the following table for the given association rules:

Rule	Support	Confidence
juice → cola		
cola → juice		
cola → beer		
beer → cola		

Then find all frequent item sets with a minimum support of 0.4.

Task 7.2

(Association analysis, Apriori). Given these three-element frequent item sets:

{i1, i2, i3},	{i1, i2, i4},	{i1, i3, i4},
{i2, i3, i4},	{i2, i3, i5},	{i3, i4, i5},
{i3, i5, i7},	{i2, i4, i5},	{i3, i6, i7}}

Which candidates of length 4 arise during the join phase? Which of these candidates are deleted in the pruning phase? Can there be candidates of length 5?

Task 7.3

(Association analysis). Suppose our given sample data has 6 attributes, each with 2 possible values. How many rules can be established if there is exactly one item on the right-hand side?

Task 7.4

(Association analysis). A restaurant keeps a list of which dishes are consumed, with each row containing one ordered menu. Help the chef find out what associations can be observed in the ordering behavior of the guests. Use the Apriori algorithm.

Soup	Salad	Stew	Steak	Pasta	Casserole	Beer	Soda	Dessert
x		x				x		x
x			x			x		x
	x			x			x	
x		x				x		
	x				x		x	
	x			x		x		
x			x			x		x
x		x				x		
x	x	x				x		
x		x					x	x

Find suitable thresholds for support and confidence.

7.2 Frequent Pattern Growth

There are several modifications of the Apriori algorithm as well as other methods for finding association rules. One such alternative method is *FP growth* (frequent pattern growth), which does not require the step of generating frequent item sets, cf. [→ HKP12]. The idea behind the method is to generate a directed graph based on the relative frequencies of the individual items by cleverly representing the item sets and counting the frequencies. The frequent item sets, referred to as *frequent patterns* in this method, can then be read from this graph.

We consider the following item sets:

ID	Item set
01	a, d, e, f, g
02	a, b, c, d, f, g
03	a, d, f
04	a, c, g
05	a, c, d, f, g
06	c, e, f

The first step consists of the following sub-steps:

1. Calculating the relative frequencies,
2. Reducing the item sets by removing the items that do not meet the support requirements, and
3. Sorting the item sets in descending order of relative frequency of the items.

The following tables show the result when we choose 40 % as the minimum support.

Item	Frequency	Support met?
a	5	yes
b	1	no
c	4	yes
d	4	yes
e	2	no
f	5	yes
g	4	yes

ID	Item set
01	a, f, d, g
02	a, f, c, d, g
03	a, f, d
04	a, c, g
05	a, f, c, d, g
06	f, c

Items b and e do not meet the support requirement and are deleted. The item sets are then sorted. These sorted item sets are referred to as *patterns*.

Now the so-called *FP tree* is constructed. All patterns (the sorted item sets) are run through and transferred to the FP tree. An additional table is created in which all remaining items are entered with a link to the first node generated. If we start by transferring the first pattern (item set 01), the result is the tree shown in → Figure 7.3. We have noted the (previous) frequency in the graph.

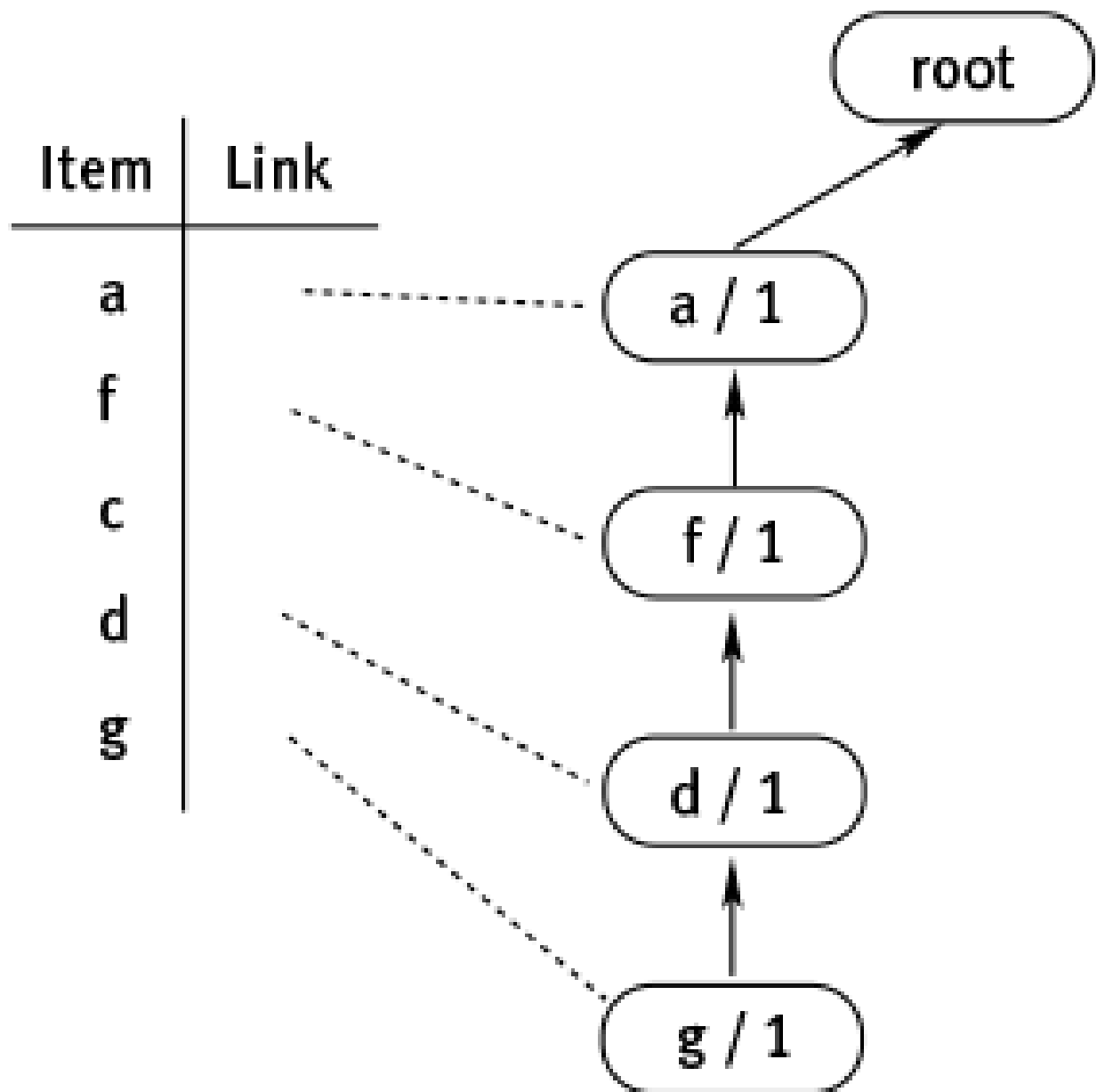


Fig. 7.3 FP tree, step 1, afdg

Now we incorporate the next pattern (item set 02), see → Figure 7.4 (left). We start at the root and go as far as possible along already known paths. There, we increment the counters by 1 each time. Only when a new subpath begins, here from item c, the current path is left. For item set 03, this path already exists in its entirety, see → Figure 7.4 (right).

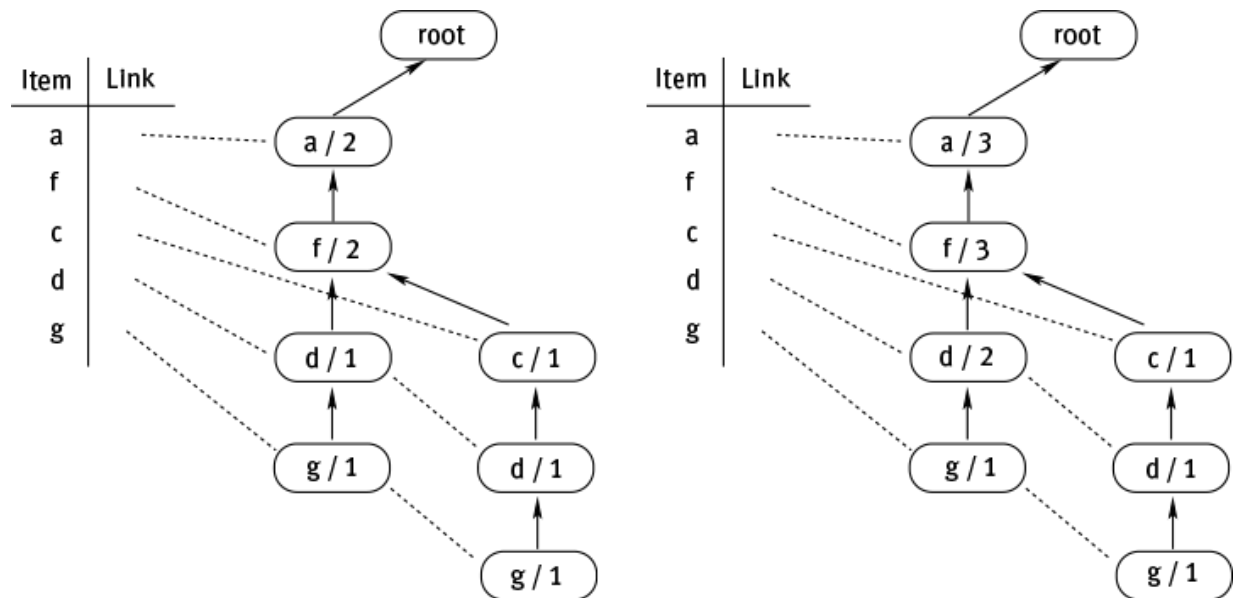


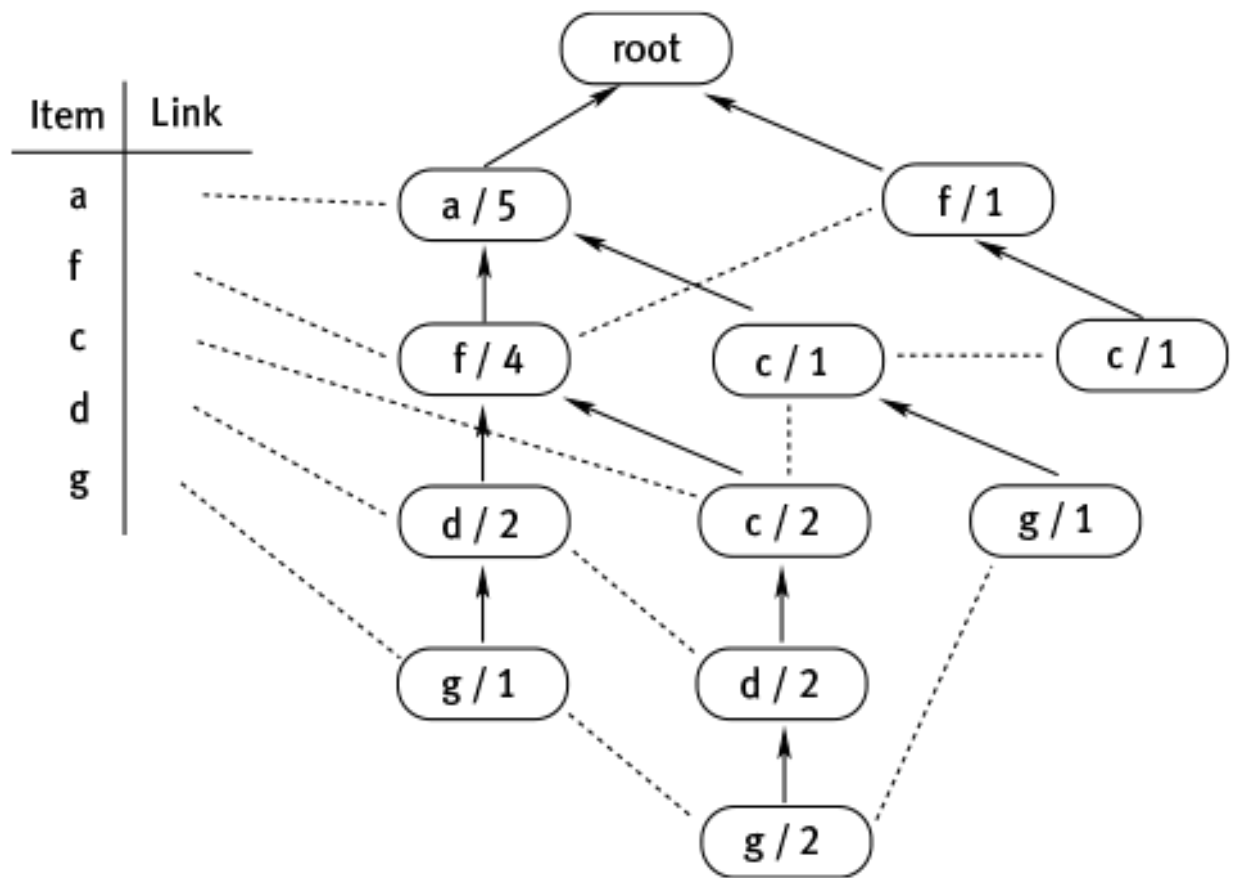
Fig. 7.4 FP tree, step 2, afcdg and step 3, afd

After processing all six patterns, we obtain the tree shown in
→ Figure 7.5.

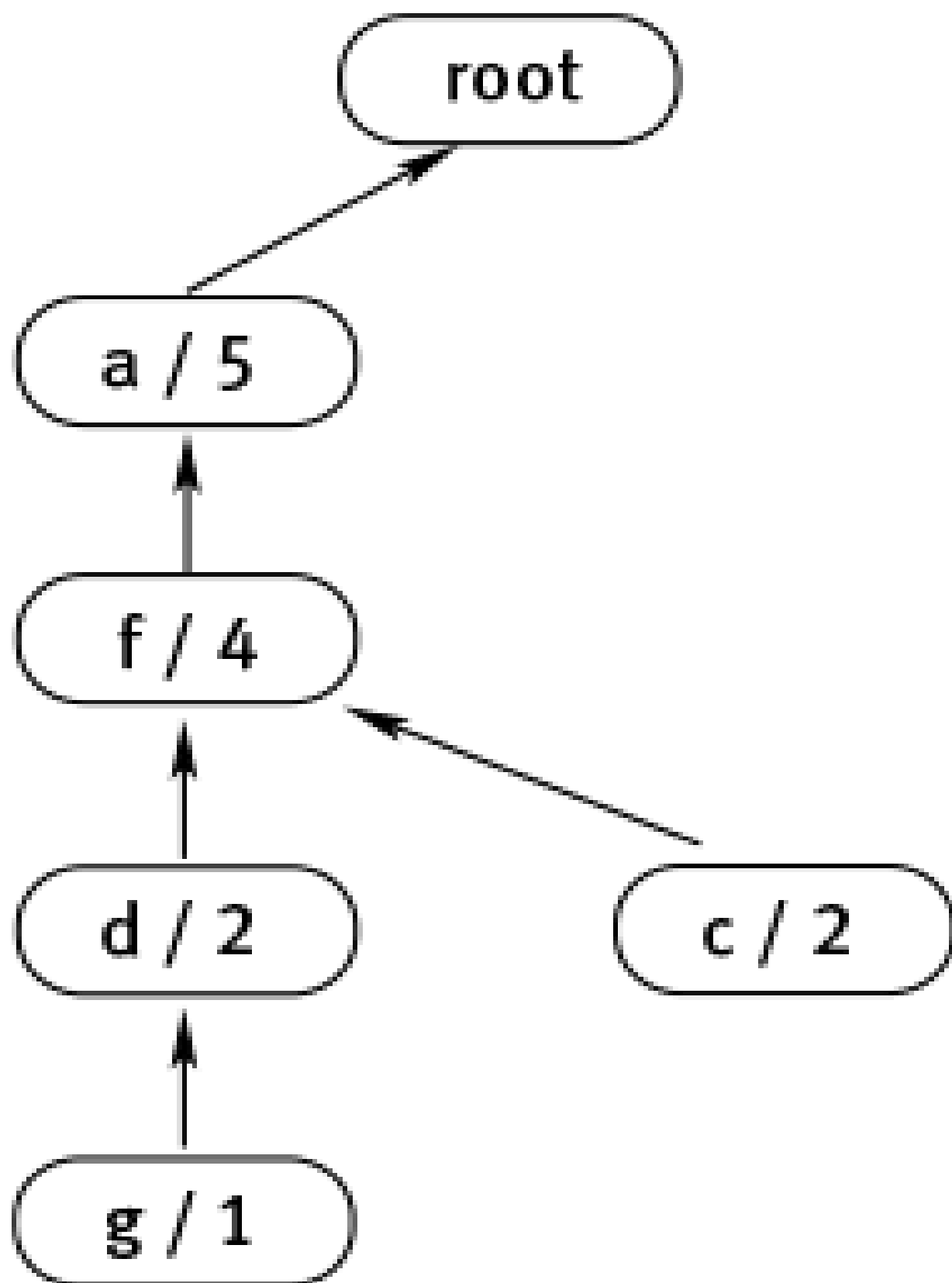
Now we need to clarify how to extract the association rules from this tree. This step is only outlined here; a detailed description can be found in [HKP12] or [→ Bra20].

In order to have access to all items, the different nodes belonging to *one* item are connected. This is represented by the dashed lines.

First, let's look at a simplified version of the FP tree, known as a *single prefix path FP tree*. The distinguishing feature of this tree is that only *one* path originates from the root, which then branches off at some point (see → Figure 7.6).



7.5 FP tree



7.6 Single prefix path FP tree

The tree consists of two parts, the prefix P (here: af) and a multipath part MP , which consists of two subpaths (dg and c). Searches can now be performed separately in both subtrees P and MP . In this case, the search is straightforward. The MP subtree does not contain any frequent item sets, as we had specified a support of 40 %.

When searching for suitable candidates, we proceed from the bottom up. For example, we see that g occurs only once and can immediately exclude g in this subtree. Now we also recognize the advantage of the sorted representation of the items. The support of a set is always determined by the lowest element of the corresponding subtree.

If there were a frequent item set in MP , it could be linked to any frequent item set from P . In the prefix part of the tree, we find three sets¹: $\{a\}$, $\{f\}$, and $\{a, f\}$. Since the MP part does not provide any candidates, the only set that contains more than one item is $\{a, f\}$.

Let's return to the FP tree from → Figure 7.5. The frequent item set cg also fulfills the support requirement. This is determined by first eliminating all paths that do not contain g (→ Figure 7.7) and then examining all other g nodes starting from g . The dashed connections are needed for this. We can ignore all nodes below g , as these are considered separately. We only work from bottom to top.

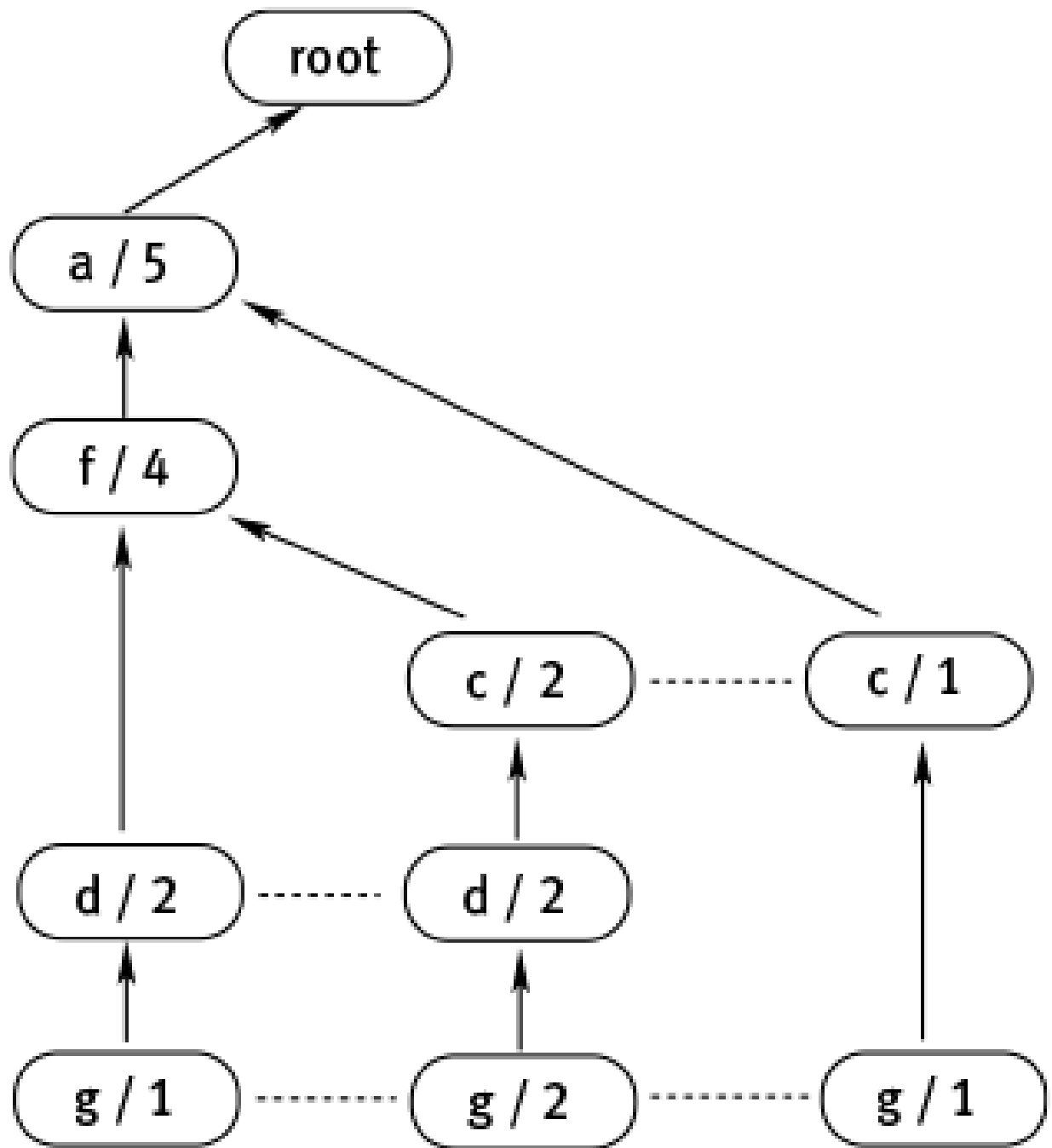


Fig. 7.7 Reduced FP tree for g

Now collect all patterns that are possible with g as a suffix:

afd/1 and afcd/2 and ac/1

These three paths do not meet the support requirement. However, we can combine them into two paths that meet the minimum support requirement of 40 %. The patterns afd and afcd have afd/3 as a common subpattern; afcd and ac become ac/3 :

afd/3 and ac/3

The patterns acg and afdg, and all their subsets, are therefore frequent patterns. This is done for all items. The results are summarized in the following table

Item	Pattern in FP tree	Reduced FP tree	Frequent Pattern
g	afd/1; afcd/2; ac/1	afd/3; ac/3	afdg, afg, adg, fdg, ag, fg, dg; acg, cg, (ag)
d	af/2; afc/2	af/4	afd, ad, fd
c	af/2; a/1; f/1	a/3; f/3	ac, fc
f	a/4	a/4	af

Only those frequent patterns that end with the respective item appear.

Taking into account the above explanations on single prefix path trees, finding all combinations can be simplified even further.

Task 7.5

(Association analysis, FP growth).

Use the FP growth method to find the frequent patterns in the movie theater data (→ Example 7.1). We require a minimum support of 50 %.

Movie theater visit ID	Visitors
k ₁	Anne, Clara, Ellen
k ₂	Anne, Ellen, Gerald
k ₃	Anne, Clara, Ellen, Frank, Gerald
k ₄	Anne, Clara, Harry
k ₅	Ben, Clara, Ellen, Frank, Gerald
k ₆	Ben, Clara, Ellen, Gerald, Harry

7.3 Association rules for special tasks

In this section, we will take a closer look at some of the special association rules mentioned in → Chapter 4. This includes hierarchical, quantitative, and temporal association rules. Other types, such as fuzzy association rules, which can handle vague terms, are not covered in this book.

7.3.1 Hierarchical association rules

Often, support must be set at an extremely low level because the variety of items and thus the heterogeneity is too great. This is very often the case with shopping basket analyses. The individual products are stored in detail in the market basket databases. The database distinguishes between a liter of milk from supplier A and a liter of milk from supplier B. This is a hindrance to market basket analysis. Here, we are interested in a general rule such as:

Those who buy milk also buy cocoa.

In this context, the distinction between manufacturers A and B is not useful. Of course, it can make sense to distinguish between suppliers. But in general, summarizing products is a good idea.

Example 7.4

(Hierarchical association rules). Let's look at a small excerpt from a fictional market basket database. Here, we have already moved away from manufacturers.

Fork	Spoon	Knife	Cup	Plate	Mug	...
X					X	...
	X			X		...
		X	X			...
X			X			...
				X	X	...
	X	X				...

A reasonable connection between the products cannot be established here. This changes when a hierarchy of concepts is introduced. For example, we use *cutlery* as the generic term for the items *fork*, *spoon*, and *knife*, and *tableware* as the generic term for *cup*, *plate*, and *mug*. Obviously, association rules for *cutlery* and *tableware* now have higher support than rules with the original products. The subterms such as *fork*, *spoon*, and *knife* are removed from the database and replaced by an attribute such as *cutlery*.

There are some algorithms that use taxonomies right when generating frequent item sets, see [→ SA95].

7.3.2 Quantitative association rules

The sample data often contains not only ordinal or nominal attributes, but also metric values, such as the number of children. A possible association rule could be:

People who have more than three children have a large car.

At first glance, our two algorithms are not suitable for attributes such as the number of children. The above algorithms work on *Boolean* attributes. They count how often which attributes are true in combination with other attributes. However, the above algorithms can also be applied to such data. If the number of values that the numerical attribute can take is small, the numerical attribute can be converted into an ordinal one. Each value is then interpreted as an ordinal value of the attribute. Of course, this no longer works if we have many values. If there are many values, two problems arise. On the one hand, the rules may be nonsensical, because in most cases it does not make sense to distinguish between a 21-year-old and a 22-year-old person. On the other hand, it may happen that the required support is not achieved because there are too few data records for the individual values of the attribute. The treatment of numerical attributes is discussed in [→ SA96]. The idea is to form intervals thoughtfully. Intervals are always combined taking into account the required support and minimum confidence. The intervals can then be interpreted as Boolean attributes. The interval *2 or 3 children* is true (1) if the person has 2 or 3 children, otherwise false (0). Similar to hierarchical association analysis, we combine attribute values, but instead using generic terms, we use intervals.

Example 7.5

(Quantitative association rules). Given the following data:

Age	Children	Married	Cars
35	3	y	2
27	1	y	1
40	4	y	2
24	1	n	0
42	4	y	2
33	2	y	2
23	0	n	0

The attribute *married* is nominal, the others are metric. The attributes *children* and *cars* can also be considered ordinal, as there are only a few possible values. If we require a support of 20 %, we always need at least two data records that contain the same value. This means that the values 0 and 2 for the number of children are excluded, as they only occur once each. The same applies to the value 1 for the number of cars.

Of course, this approach also gives us rules, for example:

IF children = 4 THEN cars = 2

IF married = n THEN cars = 0 (and vice versa)

Nevertheless, this is unsatisfactory because we have no chance of coming up with a rule that predicts a maximum of 1 for the number of children for young people.

We now define intervals suitable for *age* and *children*:

- *age*<28, *age*28-39, *age*>39
- *children*0, *children*>1

Furthermore, we transform the attributes into Boolean values, which leads to the following table.

Age <28	Age 28-39	Age >39	Children 01	Children >1	Married	Car0	Car1	Car2
0	1	0	0	1	y	0	0	1
1	0	0	1	0	y	0	1	0
0	0	1	0	1	y	0	0	1
1	0	0	1	0	n	1	0	0
0	0	1	0	1	y	0	0	1
0	1	0	0	1	y	0	0	1
1	0	0	1	0	n	1	0	0

This yields further meaningful rules. For example, the following rule is now found:

IF age<28 THEN children01

7.3.3 Generation of temporal association rules

All previous types of association rules work with information that does not take temporal dependencies into account. However, there are applications in which the temporal sequence is particularly important, for example in forecasts such as weather forecasts. In market basket analysis, it can also be useful to consider temporal dependencies. For example, it can be useful to know that one or two days after high sales of product A, product B is also purchased more frequently.

The procedure for generating association rules based on temporal sequences in a basket can be as follows:

Sorting phase

During the sorting phase, transactions are assigned to individual customers and placed in chronological order. This creates so-called customer sequences. When creating sequences, it is possible to take time frames into account. This means that if the intervals are very

large, the transactions are not considered to be one sequence; instead, several sequences are created. If the time intervals are very small, transactions are combined. For example, if a customer makes two purchases within 10 minutes, the two transactions should be counted as one, as the customer probably just forgot something and returned.

Generation phase

Here, the frequent item sets are searched for. Support is determined by the proportion of customers who include the item in any of their transactions. Support is only counted once per customer.

Transformation phase

The sequences are replaced by the frequent item sets they contain.

Sequence phase

The desired sequences are determined with the help of the frequent item sets. The process is the same as for the Apriori algorithm.

Maximum phase

The maximum sequences are determined from the set of generated item sets.

Task 7.6

(Cross selling). The following purchases are given:

Purchase ID	Purchased items
t ₁	bread, matches, cigarettes, beer
t ₂	matches, cigarettes, wine
t ₃	bread, matches, water
t ₄	cigarettes, beer, matches
t ₅	bread, matches, cigarettes, beer, wine
t ₆	water

First, complete the following table for the given association rules:

Rule	Support	Confidence
matches → cigarettes		
cigarettes → matches		
cigarettes → beer		
beer → cigarettes		

Find all frequent item sets with a minimum support of 0.4.

Task 7.7 (Survey).

On the book's website, → <https://www.fww.hs-wismar.de/datamining/>, you will find the results of a survey. Both the questionnaire and the collected data are stored there in csv format. Search for association rules in this survey data.

8 Data preparation

Experience is something you don't get until just after you need it.

Olivier's Law

So far, we have avoided what is probably the most difficult task in practice: *data preparation*. We will look at the KDD process (*Knowledge Discovery in Databases*) as introduced in → Section 1.2 and summarize the first three sub-processes for *data preparation*:

1. Data selection
2. Data preprocessing
3. Data transformation

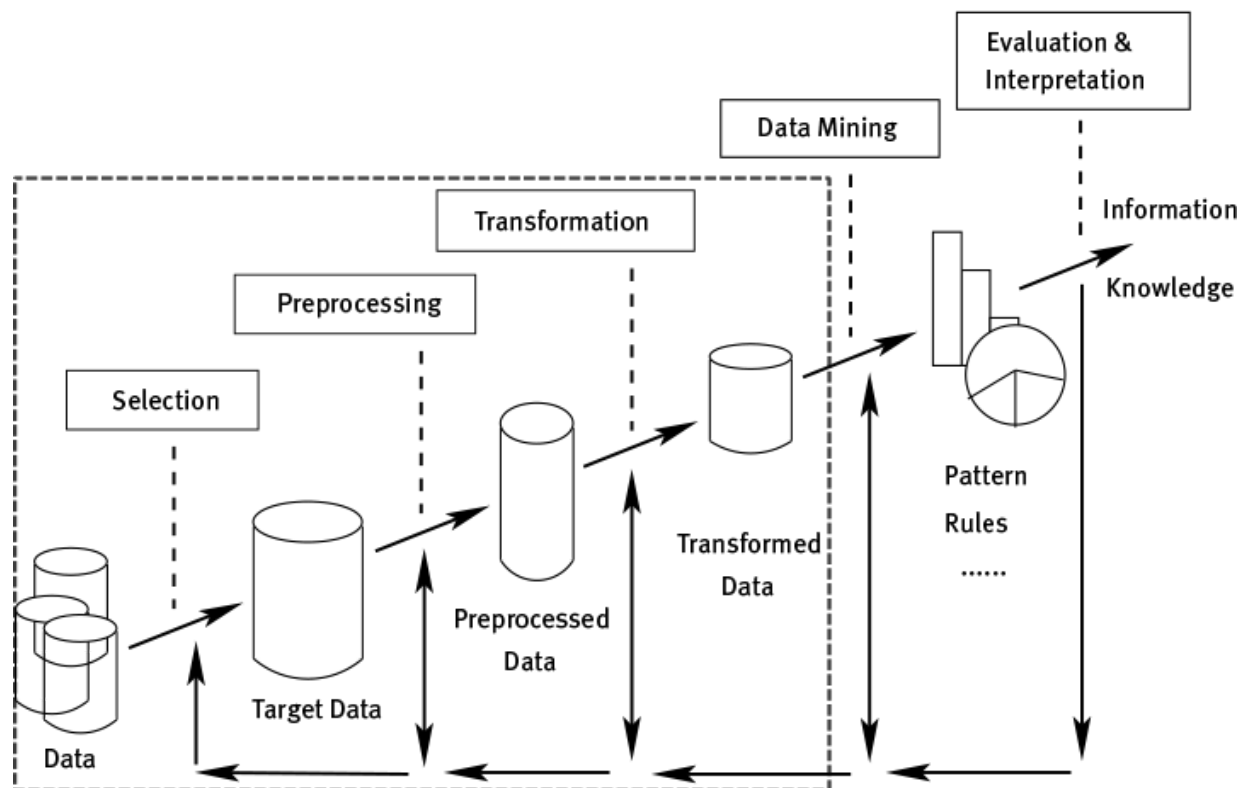


Fig. 8.1 Data preparation (own representation based on [→ FPSS96])

8.1 Motivation

In most cases, data cannot be used directly for a data mining method, but must first be prepared accordingly. To do this, it is necessary to examine the specific data available. The first step is to collect all existing data. The data may not be stored in a *single* data table, but may need to be compiled from several databases or other sources.

In addition, the data is rarely available in the desired or required quality. The quality of data can be assessed using the following criteria and questions:

Comprehensibility: Is the data comprehensible? Can we interpret it?

Usefulness: Is the data valuable for the intended purpose?

Validity: Is the data valid? Are the sources verifiable?

Credibility: Does the data match our experience?

Accuracy: Is the data accurate? Is it consistent and complete?

Timeliness: Is the data up to date?

Volatility: The available data should be representative over a certain period of time. It does not make sense to use data that is subject to extreme fluctuations over time.

These criteria are largely at the level of *business understanding* (see → Figure 1.2, p. 4). When we approach the actual data analysis, we move to the level of *data understanding*. The following tasks and questions arise with regard to data quality:

- All values must be considered and checked. Are they reasonable and what do they mean?
- Does the data cover the world to be analyzed in a representative manner? It is difficult or even impossible to perform analyses if significant attributes do not exhibit their entire range of values in the given sample set. If only data on women is available, it will not be possible to make general statements about all people.
- Is the data faulty?
- Are there missing values in the attributes? If so, could these missing values contain information?
- Is the data plausible, or is it contradictory?

We will explore these questions and suggest possible solutions. Before systematically examining the individual steps involved in preparing data, we will address a few problems to illustrate the necessity of data preparation.

Problems in data analysis are caused, for example, by *outliers*, i.e., data points that deviate significantly from the other data or lie outside the usual range of values. It therefore makes sense to examine metric attributes more closely in terms of their range of value. If data is available for people aged 30 to 50, a data record on a 90-year-old is problematic on the one hand, it may be an outlier, and on the other hand, the 90 could simply be an incorrect value. Whether such outliers should be hidden or adapted for data mining, or whether it is better to use them in their original form, depends on the specific context.

Another problem area in data preparation arises from *missing*, *inaccurate*, *incorrect*, or *contradictory* values. There are many reasons for missing or poor-quality data. In a survey, people may refuse to answer for personal reasons or may consciously or unconsciously provide false information.

If the structure of a database is changed but the data records are not adjusted accordingly, this can lead to missing attributes.

And, of course, the possibility of careless mistakes (for example, when entering values manually) must always be taken into account.

Possible types of errors are summarized in → Table 8.1 (cf. → Section 8.3).

Tab. 8.1 Typical data errors

Pers. no.	Name	Gender	Date of birth	Age	Postal code	Street	House no.
11	Smith, Thomas	m	06/12/1977	68	23966	Wagner Street	0
23	Thomas Smith	m	00/00/0000	zero	23966	Wagner Street	4
11	Miller, Rosie	f	12/16/1986	39	06119	Mozart Street	7
42	Warren, Anne	fem	02/30/1978	47	66125	Bach Street	31

- Personnel number 11 was assigned twice.

- The data in lines 1 and 2 most likely describe the same person, as suggested by the name (despite the different order of first and last names) and the identical street name.
- In line 1, the date of birth and age contradict each other.
- Three values are missing (a default value is specified in each case): in line 1, the house number 0, and in line 2, the date of birth and the age zero.
- The postal code 06119 does not exist in Germany (to our knowledge).
- The entry *fem* is not a valid value for gender.
- Ms. Warren's date of birth cannot be correct.

In addition to problems with the values themselves, difficulties may arise in processing them. If computing capacity is limited, so-called *dimension reduction* may help. On the one hand, a database contains many attributes for a single data record. In data mining, this can lead to problems with high dimensionality, which always involves high computing costs and thus increased program runtimes. However, these are sometimes unacceptable.

On the other hand, human visual perception is limited. Three-dimensional visualization or even four-dimensional space-time visualization presents us with major challenges. Since visualization is used not only after data analysis to illustrate the results, but also *before* data analysis, dimension reduction is already useful at this stage.

Another problem in data preparation arises from the fact that data is available in formats that are unsuitable for the respective data mining process. If, for example, certain information is available in kilometers and other information in miles, data transformation is necessary. It may also be necessary to scale the data.

Data preparation plays a major role in data mining, as the quality of the data is crucial not only for the runtime behavior of the data analysis, but also for the quality of the results. Data mining is an iterative process: experiments are carried out, a result is calculated and checked. It is very rare for the first run to be successful. In our experience, around 80 % of the effort involved in a data mining project is spent on data preparation.

There is one simple reason why data preparation is crucial for the quality of the results: if the data is poor or poorly prepared, the process cannot be expected to iron out the problems and deliver good results. A general principle of computing also applies here: garbage in, garbage out.

In the following sections, we will take a closer look at the typical problems mentioned above and present approaches for solving them. We will discuss a number of techniques, but we cannot present any magic formulas, as every data mining project is different. This is why the *data understanding* phase (cf. CRISP-DM model, → Figure 1.2, p. 4) is so important. Only with an *understanding* of the data it is possible to prepare the data in a meaningful way.

However, before starting with data preparation, you should use simple *statistical analyses* to get an impression of the available data. For example, the *number of missing values* is of interest for all characteristics. For numerical attributes, the values for *mean*, *median*, *standard deviation*, *maximum*, and *minimum* provide a good insight into the data. For nominal and ordinal data, information about the *possible values* and their *frequency* helps us to understand the data. In KNIME, the first step after importing the data should be to connect the *Statistics* node.

Once you have gained an impression of the data through statistical analysis, you can begin with data preparation.

8.2 Types of data preparation

Before we address the individual problems and possible solution techniques, let's first look at the phases of data preparation. We distinguish between the following steps:

Data selection and data integration

In this step, suitable or necessary data is selected and combined. Data from different sources is merged into a *single* database table.

Data cleaning

The data is cleaned up. Incorrect, inconsistent, or incomplete data points are identified so that they can be corrected or removed from the data set.

Data reduction

The data is reduced, for example in terms of dimension.

Data transformation

The data is adapted to the selected analysis method, whereby it is converted, normalized, or coded.

Data cleaning and data reduction are both part of the second step in the KDD process (→ Figure 8.1), the data preprocessing. However, since they are different in nature, we will treat them separately in this chapter. Data cleaning and data reduction are largely *method-independent*, but the subsequent data transformation does *depend* on the method used.

8.2.1 Data selection and data integration

Before you can start a data mining project, you need to provide or select data. This step is called *data selection*. Since the selected data sometimes comes from different tables or even different databases, it must then be merged into a single data table, which is called *data integration*. This can involve some difficulties:

- Even within a *single* company, each branch may operate its own database.
- Databases can have different structures.
- The syntax or semantics of attributes that are essentially the same may differ.

The purpose of data integration is to merge data from multiple data sets from different sources. Ideally, the result is a consistent table with *coherent data records*. The following problems must be solved during selection and integration:

Entity identification problem Which characteristics have the same semantics? If, for example, two attributes *customer_ID* and *customer_number* exist, the question arises as to whether both attributes mean the same thing. The so-called metadata, data that describes the data, may provide information.

Redundancies Redundancies can arise from inconsistencies in the nomenclature of attributes or dimensions. For example, the two attributes *Name* and *name* are syntactically different, but probably have the same semantics.

Contradictions When combining different data sources, contradictions in the data may occur. For example, there may be several addresses for one person.

Data value conflict The data records of an attribute may have different units of measurement, for example, distances given in miles

or kilometers.

Violation of referential integrity Referential integrity can be violated by a foreign key reference to a non-existent key value in another table.

Example 8.1

(Violation of referential integrity). The following data is given:

Customer data			Orders	
Customer no.	Name	City	Order no.	Customer no.
1	Miller	Berlin	1	2
2	Brown	Munich	3	1
3	Lehmann	Hamburg	2	1
			4	4

The orders table refers to customer 4, who does not exist.

The difficulties mentioned arise from the merging of data from different sources. In the following sections, we assume that all data is available in a *single* data table.

8.2.2 Data cleaning

Data in the real world is sometimes incomplete, contains errors or outliers, or is even inconsistent. These problems need to be addressed, as incomplete data can lead to errors in the data mining process, resulting in unreliable or even incorrect results.

When cleaning the data, care must be taken to ensure that, as far as possible, no new information is added. Inserted values or data should be information-neutral, meaning that the existing information originating from the real world should not be distorted. Neutrality should always be the goal, although it may not always be achievable.

We will address the following issues:

- Missing values

- Noisy data
- Incorrect data
- Inconsistent data

Missing values

If values are missing, it must first be clarified whether this is an error, or whether information can be derived from the absence of one or more values. If someone does not fill in certain fields on a loan application, this may be intentional, they may be reluctant to disclose information, or it may be unintentional. Here, we focus on eliminating genuine errors.

Missing values can be handled as follows:

- *Ignore attribute*
The attribute is removed completely, i.e., an entire column is deleted from the data table. Caution: If you delete the entire column, information may be lost. An attribute may only be deleted if there are still enough other (*complete*) attributes available.
- *Insert missing values manually*
Manual insertion is very time-consuming, especially with large amounts of data, and therefore unrealistic. For example, in the data set of the Data Mining Cup 2001, the values for the attribute *customer_since* are missing for approximately 500 customers in the training data and approximately 1000 customers in the test data. In this case, the values cannot be added manually, but must be added automatically.
- *Global constant*
Missing values are replaced by a global constant, such as *unknown* or *minus_infinity*. This method is useful if an empty field is considered information or if many values are missing.
- *Average value*
For metric attributes, it is possible to use the *average* value or the *median* of all entries.
For classification tasks, this approach can be refined by using only data records from the same class to calculate the average value. This is a simple way to clean up data. It is useful if the (numerical) values of the class are close together and it can be assumed that the missing

values also lie within this range. Adding average values can be implemented quickly.

If no classes are available, the average value of the data records that are similar to the current data record can be used. This approach follows the idea of *k-nearest neighbor*.

- *Most probable value*

Another option is to replace the missing values with the most probable value for this attribute, which can be determined using statistical methods, for example. This option should be chosen if there is no value available for one or a few cases and there is sufficient evidence for a reasonable or justified value.

- *Most frequent value*

If the attribute is non-numerical, the most frequent value (the so-called modal value) can be used.

- *Relationship between attributes*

If there is a relationship between the attribute with missing values and another attribute, this can be exploited. For example, if a person's age is missing, it can be calculated from the *year of birth* attribute.

If a *regression analysis* finds a correlation between numerical attributes, the missing values can be calculated using the regression function.

Association analysis techniques can also be used to establish relationships between attributes and predict missing attribute values. Another option is to use classification methods to predict the missing value. The attribute with the missing values temporarily becomes the target attribute for this purpose. Then, for example, the k-nearest neighbor method is used to predict the missing values.

- *Mark data record as incorrect*

As a fallback solution, we can still exclude the relevant data records from further processing. This is useful if *sufficient complete* data is available. However, this is only a solution for the training data; in practical application, missing values must still be addressed.

KNIME offers many ways to handle missing values. For integer values, these are shown in the screenshot in → Figure 8.2.

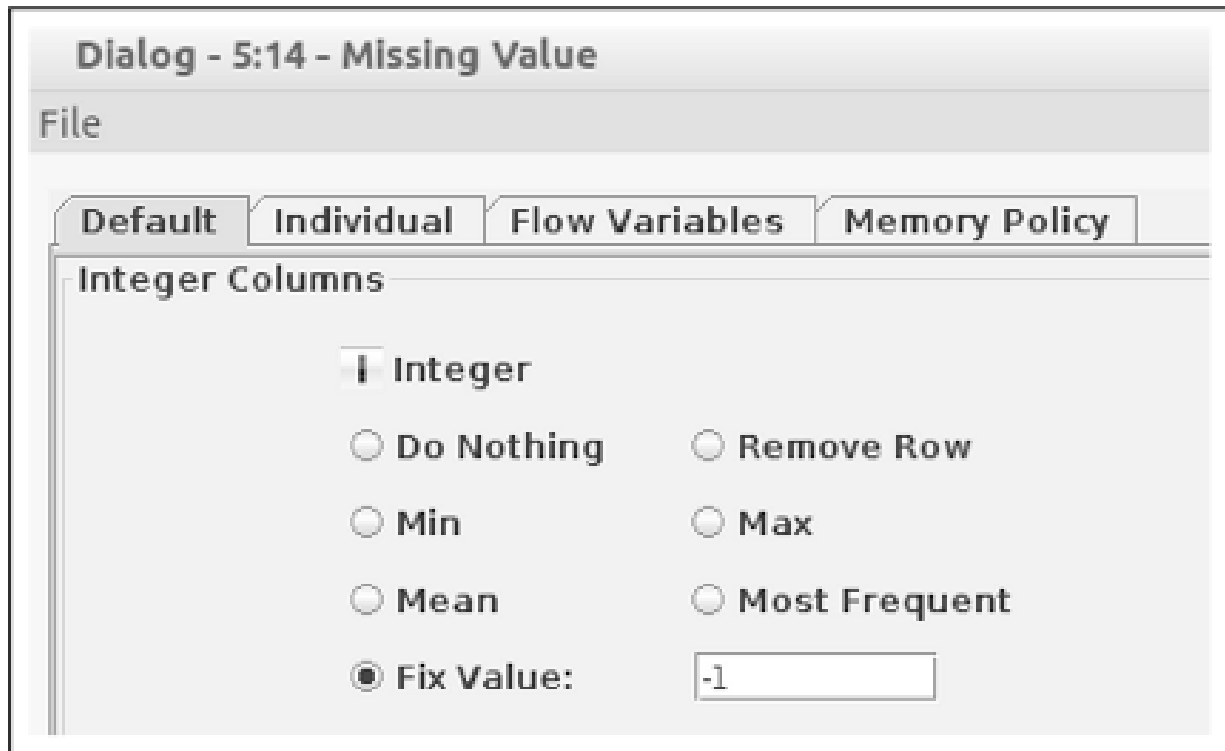


Fig. 8.2 Handling missing values in KNIME (default settings)

Accordingly, there are several options for real numbers. Missing character strings can be replaced by the most frequent or a fixed value. In addition, KNIME allows you to specify individually for each attribute how missing values should be handled, see → Figure 8.3.

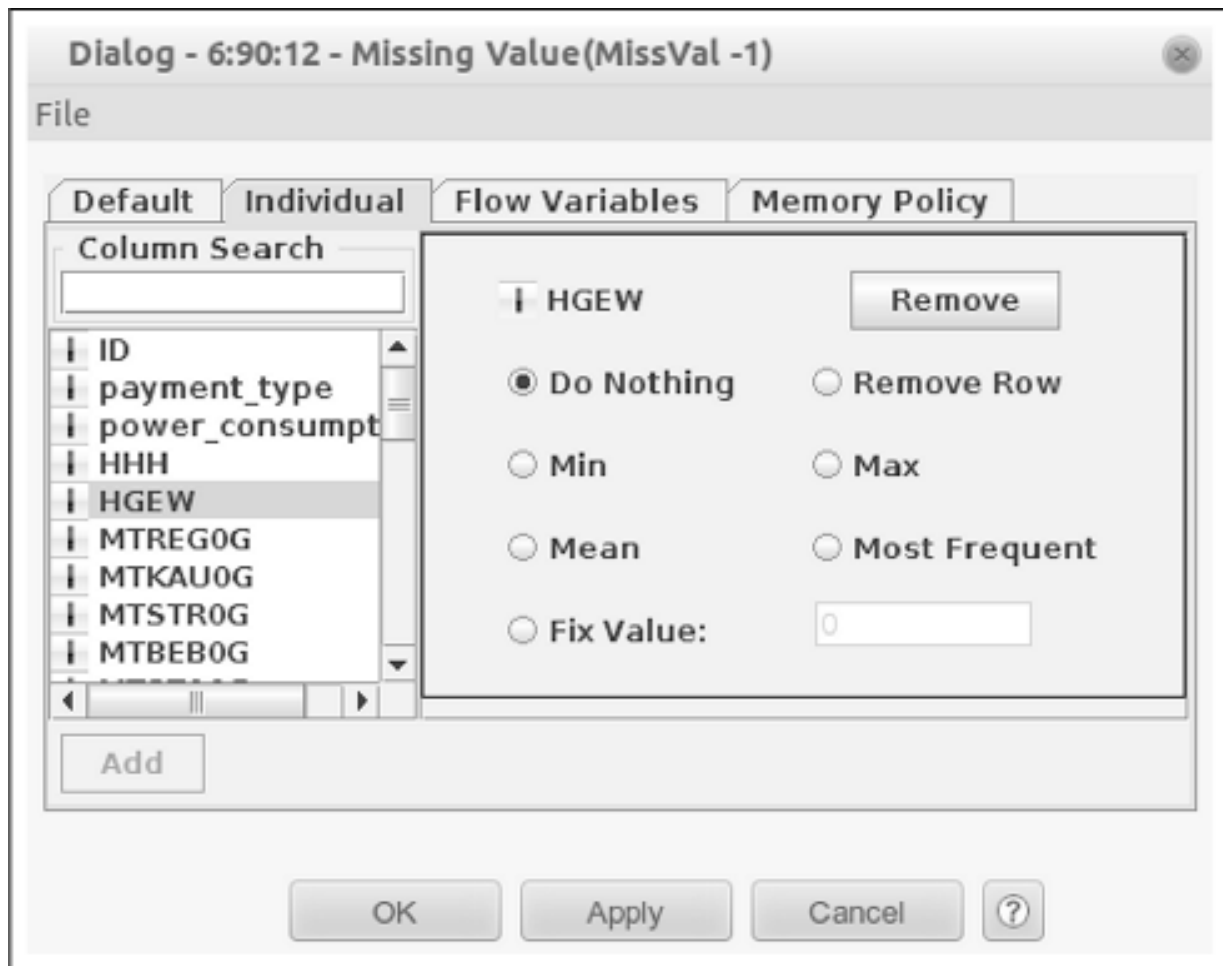


Fig. 8.3 Handling missing values in KNIME (individual settings)

The techniques for replacing missing values are divided into three categories:

MCAR ... Missing completely at random The value used is generated randomly without restriction. It must not depend on any other attribute or on other values of the current attribute. Restrictions that apply to the attribute are of course taken into account; for example a random value of 300 for age is obviously nonsensical.

MAR ... Missing at random Here, it is acceptable to include the other attributes in the generation of a suitable value for the current attribute. However, the values of the current attribute are not included.

MNAR ... Missing not at random Here, the generation of a suitable value is entirely attribute-specific.

If someone does not disclose their attribute value, this is often due to the attribute itself. For example in the case of salary the respondent simply does not want his income to be known. When inserting a value for the missing value, we therefore refer directly to the attribute *salary*. This is part of MNAR. If someone has not disclosed their educational qualifications, we can include the values of other attributes (MAR): for example, if someone is a lawyer, then they have studied.

With these techniques, we are forced to abandon our goal of modifying our database in an *information-neutral* manner. Inserting values, whatever method is used, leads to *changes in the database* and can affect the quality of the data. The *semantics of the data* can also be distorted. Ignoring or omitting attributes or data records distorts the data with regard to the probability distribution of the attribute values.

We violate our goal that any change to our database should be information-neutral. However, when values are missing, we have no other choice. All changes must be documented in detail and in a comprehensible manner, not least in order to be able to assess their impact on the result.

In statistics, the treatment of *missing values*, or more precisely, the replacement of missing values with meaningful values, is referred to as *imputation*.

Noisy data and outliers

By noisy data, we mean data that contains minor errors. Such errors arise from inaccurate measurements or estimates. The data can be smoothed (adjusted) in various ways:

- *Binning* The noisy data is grouped and replaced by mean values or threshold values. KNIME offers several methods for binning. The following workflow performs binning for the Iris data (see → Appendix A.1 and → Example 6.1).

ARFF Reader

Auto-Binner



Read Iris Data

5 Groups

On the one hand, binning can be left entirely to KNIME by specifying the desired number of bins:

sepal length	sepal width	petal length	petal width	class	sepal length binned	sepal width binned	petal length binned	petal width binned
5.1	3.5	1.4	0.2	Iris-setosa	bin 2	bin 4	bin 1	bin 1
4.9	3	1.4	0.2	Iris-setosa	bin 1	bin 3	bin 1	bin 1
4.7	3.2	1.3	0.2	Iris-setosa	bin 1	bin 3	bin 1	bin 1
4.6	3.1	1.5	0.2	Iris-setosa	bin 1	bin 3	bin 1	bin 1
5	3.6	1.4	0.2	Iris-setosa	bin 1	bin 4	bin 1	bin 1
5.4	3.9	1.7	0.4	Iris-setosa	bin 2	bin 4	bin 1	bin 1
4.6	3.4	1.4	0.3	Iris-setosa	bin 1	bin 3	bin 1	bin 1
5	3.4	1.5	0.2	Iris-setosa	bin 1	bin 3	bin 1	bin 1
4.4	2.9	1.4	0.2	Iris-setosa	bin 1	bin 2	bin 1	bin 1
...								

On the other hand, the intervals can also be set manually. In KNIME, this is done with the *Numeric Binner*.

- *Regression* The data is described by a mathematical function. Then the real, noisy data values are replaced by the calculated values of the function found using linear regression (see → Section 3.3).
- *Clustering* Outliers can be identified by forming clusters. Clusters of similar values are formed, for example using a density-based method such as *DBScan* (→ Section 6.6). The outliers are then located outside the clusters.
- *Combined machine/human inspection* The computer generates a list of (apparently) strange values, after which humans filter out the outliers based on experience.

For further methods of detecting outliers, we refer to the literature, for example [SMA19]. Once outliers have been detected, they must be dealt with. The same options are available for this as for handling missing values.

Inconsistent and incorrect data

There are many possible errors and causes of errors. We have already discussed some errors and sources of errors in the previous section on data selection and data integration.

Errors can occur at the structural level, as described in the section on data integration. However, errors can also arise from human error, such as incorrect entries or typos. Cryptic entries, such as abbreviations, can also cause problems.

Values may also fall outside their value range, and values may occur that are not plausible.

Example 8.2

(Incorrect and invalid values).

- Violated value range: If we represent months by integer numbers $1 \leq x \leq 12$, then 14 is not a valid value.
- Violation of plausibility: A customer who otherwise has low sales suddenly has very high annual sales.

Contradictory data can arise, for example, when attribute values do not match.

Example 8.3

(Contradictory data).

- The customer's age does not match their year of birth.
- A person's place of residence and postal code do not match.

Multiple occurrences of data records can also cause problems, as these records are given greater weight in many processes due to their multiple occurrences. This occurs, for example, in decision tree learning, in training neural networks, and the k-nearest neighbor method.

How can such problems be solved? In most cases, manual correction is required in these situations, or a preprocessing program specifically tailored to the problem at hand must be developed.

In general, there are two categories of corrections for such errors:

- *Deleting*
We can delete the data record with the incorrect value. We can also remove attributes that contain multiple incorrect values.
- *Using other data records*
We try to find a plausible value based on the error-free data records.

The first option is obvious, but (similar to the treatment of missing values) has the disadvantage that our data set may shrink too much, making meaningful data mining impossible. The second option (using other data records) is also easy to understand. Both options follow the procedure for dealing with missing values.

Problems arise when we have conflicting values and cannot determine which value is incorrect and which is correct. Of course, it is still possible to delete the data, but this immediately affects at least two data records.

8.2.3 Data reduction

In many data mining tasks, the data sets are very large. This can make data mining with these amounts of data very time-consuming or even impossible. The aim of data reduction is to alleviate this problem. There are

two options for this. On the one hand, complexity can be reduced by combining attributes. On the other hand, the size of the problem can be reduced by taking a suitable sample, a representative subset of all given data records. Techniques for data reduction include:

1. Aggregation
2. Dimension reduction
3. Data compression
4. Numerical data reduction

Aggregation

Aggregation refers to the *combining* of facts into a *single fact* or the generalization of data. Data can thus be replaced by its mean values or partial values can be combined into a total sum (see → Section 8.2.5, p. 220). If a company's monthly sales figures are available, these values can be added up to give the annual sales figure. Instead of 12 individual data records, only one data record is then used for further processing.

In addition to this row-by-row aggregation, column-by-column aggregation is also possible, which aggregates multiple attributes. If *day*, *month*, and *year* are given as individual attributes, these three attributes can be combined into one attribute: the *date*. Similarly, the sales figures for the individual subdivisions can be combined into one attribute, the total sales. This does not reduce the number of data records, but it does reduce the number of attributes, i.e., it achieves a *dimension reduction*.

Dimension reduction

In dimension reduction, irrelevant attributes are excluded and relevant attributes are included. The following procedures are available:

- Stepwise *forward selection*: Attributes that are obviously important are successively incorporated into the target set, with the attribute set growing accordingly.
- Stepwise *backward elimination*: Starting from the total set, attributes that appear less relevant are gradually eliminated.

Both approaches can be combined.

In the framework of dimension reduction, the question arises as to which attributes can be omitted immediately without losing any potentially significant information. Two situations come into consideration here:

1. The values of an attribute are all identical in our data. For example, if we only have data about women, we can delete the *gender* attribute. It cannot contribute anything. Of course, this changes immediately when data about men is added.
2. If all values of a nominal attribute are different, we cannot expect this attribute to contribute to the generation of a decision tree. However, if k-nearest neighbor is used (see → Section 5.1), a new data record may contain a value that has only occurred once before, which then contributes to the classification.

Techniques such as *principal component analysis* (PCA) (pp. 221) or multidimensional scaling (see → Section 8.2.5) can be applied to metric data to reduce the dimension.

Data compression

The data is either transformed or encoded to shrink the amount of data and thus reduce the complexity of data mining. The focus here is on decreasing the number of attributes. For example, binary attributes are combined into a single byte, or the separate attributes for the products *spoon*, *fork*, *knife*, etc. are combined into a single attribute called *cutlery* (see Aggregation, p. 220).

Numerical data reduction

Numerical data reduction, the selection of a representative subset of the given data records, can be achieved on the basis of random sampling. Data mining does not use the entire data set, but only a sample, which is of course much smaller. The question of how this sample is formed needs to be clarified.

- *Random sample*
Data records are selected from the source data set completely at random.
- *Representative sample*

Data records are selected randomly from the source data set, but in such a way that the sample is representative. To this end, the selection is made taking into account the frequency distributions of certain attributes. It is also desirable that each attribute value (for nominal and ordinal attributes) occurs at least once. In classification problems, each class must be sufficiently represented.

- *Stratified sample*
As with representative sampling, data records are selected at random. The sample set is divided into meaningful groups, known as strata, for example according to salary (*low, medium, high*) or age (*young, middle-aged, old*). The sample is then selected in such a way that each stratum is represented to an appropriate extent.
- *Incremental sampling*
An initial sample is expanded step by step after data analysis. This can be done using one of the above methods.
- *Average sampling*
The source data set is divided into parts and each part is analyzed independently of the others. The results are then averaged and combined to form an overall result.
- *Selective sampling*
Unproductive data records are filtered out. A sample is then taken.
- *Windowing*
Similar to incremental sampling, we start with an initial sample. This is gradually expanded to include only productive data records. At the beginning, only part of the training data (so-called data windows) is used. Then, only those training data records are added to the training set for which the method (for example, k-nearest neighbor) makes an incorrect classification.
- *Cluster-based sampling*
In this method, similar data records are grouped into clusters and one (or several) representative(s) is/are selected, which is/are then used for the analysis.

Alternatively, numerical data reduction can also be achieved using linear regression. This involves the data replacing by the coefficients of a linear regression function.

KNIME supports the sampling process. The simplest variant is shown in → Figure 8.4. The *partitioning* node offers the option of dividing the entire

data set into percentages.

For classification tasks, *Stratified sampling* should be selected in order to obtain the same frequency distribution for the target attribute in the training and validation sets as in the total set. In order to provide meaningful predictions, a method for generating a classification model requires sufficient examples for *each* class to be predicted. A prediction for good and poor customers won't work if only poor customers are present in the training set.

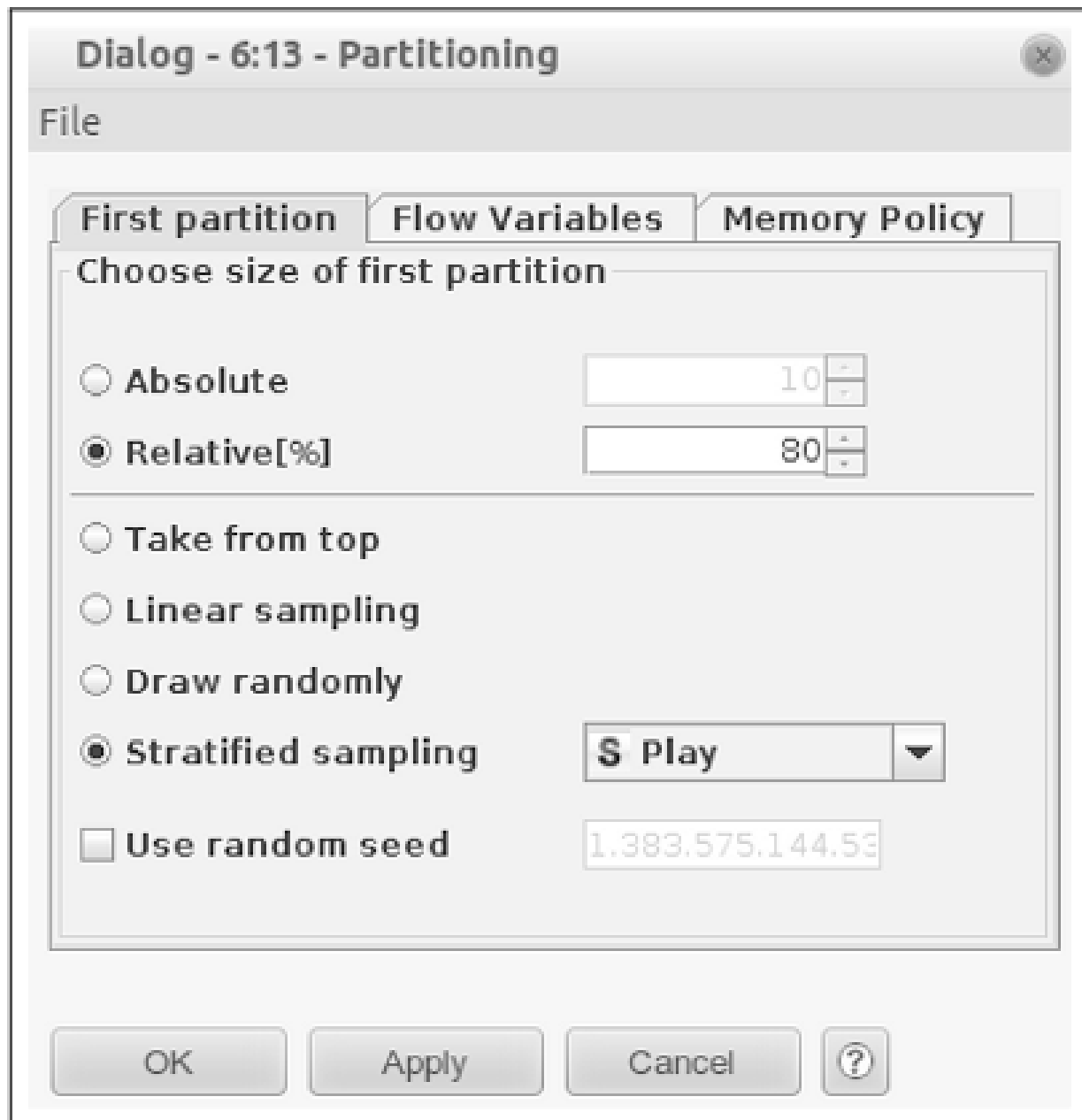


Fig. 8.4 Sample selection by percentage

8.2.4 Uneven distribution of the target attribute

In classification tasks, the problem often arises that the target classes are not represented with equal frequency in the data set. Equal frequency of target classes can be achieved by taking the class with the lowest occurrence as a measure and reducing the sample data for the other classes accordingly. This approach is known as *undersampling*. Alternatively,

the data can be filled in until all classes occur with equal frequency. This approach falls into the category of *oversampling*.

Let's consider the task discussed in detail in → Chapter 10, which involves developing a prediction model for a mailing campaign by an energy supplier. It is known that 10 % of customers cancel their subscriptions, which is reflected in the training data. In KNIME, the *Equal Size Sampling* node would drastically reduce the 90 % of non-canceling customers until there is an equal amount of data for canceling and non-canceling customers. Alternatively, the cancelers can be multiplied, see → Figure 8.5.

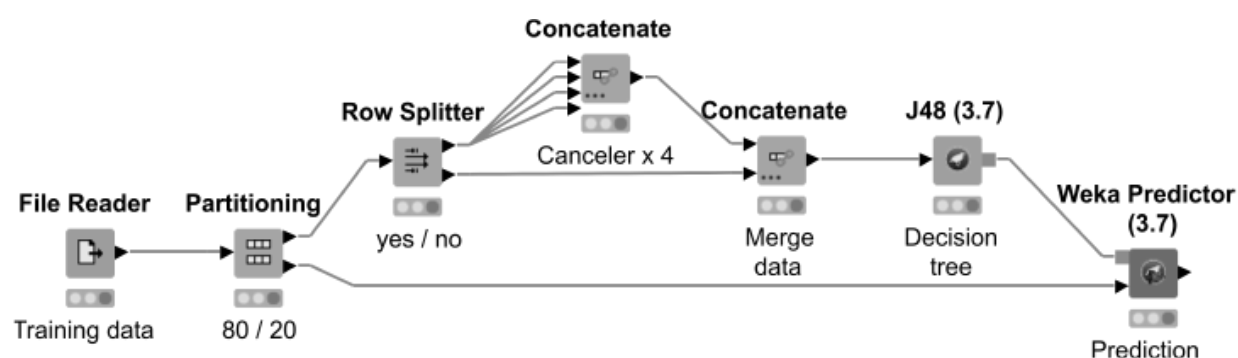


Fig. 8.5 Data multiplication in KNIME

To do this, we first separate *cancelers* from *non-cancelers* (row splitter), then we multiply the cancelers using the *Concatenate* node; in → Figure 8.5, we multiply the cancelers by four. We then concatenate the cancelers and the non-cancelers. However, this may only be done on the training set. The test set remains unchanged in order to obtain realistic results from our experiments.

A similar technique is the *Synthetic Minority Oversampling Technique* (SMOTE), which generates new artificial data records that resemble the data of the underrepresented class.

However, multiplying the number of cancelers is not without its problems, as we are changing the relative frequencies in the training set. In a method such as naive Bayes, which is based directly on relative frequencies, multiplying is therefore questionable. Multiplying should also be used with caution in k-nearest neighbor. For example, if we triple the number of cancelers, then kNN with $k = 4$ only raises the question of

whether there is a canceler among the next *two* neighbors. So if we multiply data, we have to think about a suitable *k*.

8.2.5 Data transformation

So far, we have discussed data preparation issues that arise regardless of the specific method used. In this section, we will take a closer look at the actual data mining step, i.e., data analysis in the narrower sense. Often, the data in its original form is not suitable for a specific data mining method. Data transformation has the task of converting the data into a form that can be used by the respective method.

There are a number of transformations that are used depending on the method, but there are also some that are method-independent.

Adjustments may be necessary, for example, for

1. *Data types,*
2. *Conversions or encodings,*
3. *Character strings,*
4. *Date information,*
5. *Units of measurement and scaling.*

Other transformations we have already mentioned are

1. *Combination or separation of attributes*
2. *Calculation of derived values*
3. *Data aggregation*
4. *Data smoothing*

These transformations are method-independent; they serve solely to create a consistent database.

Adjustment of data types and conversions

The question often arises as to what data type an attribute is. Is a "2" as a single value of the data type *character* or *integer*? Should a conversion be performed?

Values can be represented in different ways. To specify the days of the week, for example, we can use the character strings *Mon... Sun* or the numbers *1... 7*.

Adjustment of character strings

Can the data mining program handle *special characters*, *upper and lower case* letters, and *spaces* in the data records, or does the data need to be converted? If necessary, this must be done during data integration.

Adjusting date information and units of measurement

Dates are subject to different regulations in different countries and are therefore *coded differently*: 12.03.2003, 12-03-03, or 03-12-03.

Standardization is necessary. Data from different *time zones* may also be present and must be adjusted, otherwise the result will be distorted.

The same applies to the adjustment of units of measurement, as these often only comply with *national standards*, such as inches, degrees Celsius, or degrees Fahrenheit. The adjustment of time or date specifications and the introduction of consistent units of measurement should already be carried out during data integration.

The converting techniques considered so far are essentially *method-independent*. Often, the method used forces us to convert data into other data types. As already mentioned, the ID3 algorithm (→ Section 5.2.3) cannot handle metric data and therefore requires nominal or ordinal attributes. The same applies to the naive Bayes algorithm. In contrast, the k-nearest neighbor algorithm (→ Section 5.1) requires metric data. It may therefore be necessary to convert metric attributes into nominal attributes or vice versa.

Conversion of nominal and ordinal data into metric data

Example 8.4

(Encoding, performance classes of cars). If we have an attribute *speed* with ordinal values *very fast*, *fast*, *medium fast*, *slow*, and want to apply the k-nearest neighbor algorithm with Euclidean distance, we must convert the attribute into a numerical one. If we want to obtain normalized values between 0 and 1, we can use the following encoding:

- *very fast* → 1

- *fast* → 0.66
- *medium fast* → 0.33
- *slow* → 0

The conversion of *very fast* to 1 and *slow* to 0 is obvious, since we want to achieve an encoding in the interval [0, 1]. However, we could also encode the other two values alternatively:

- *very fast* → 1
- *fast* → 0.8
- *medium fast* → 0.5
- *slow* → 0

The only important thing is that the order relation

$$very\ fast > fast > medium\ fast > slow$$

is not violated, as the original data type is *ordinal*. Both encodings are correct; they preserve the order of the values. However, they can lead to different results in the k-nearest neighbor algorithm, as the distances between the speed values are different. In the first variant, the distance between *very fast* and *medium fast* is 0.67, while in the second it is 0.5. This can have an impact on the result.

But what if you don't have ordinal data, but nominal data? In that case, you cannot use the above conversion, because if you simply encode the colors *black*, *green*, and *blue* as 0, 0.5, and 1, you are introducing an order on the data that is not justified. The data type is not ordinal. This is where *binary encoding* helps us. Especially for neural networks, this conversion is a key form of encoding. Binary encoding turns attributes with a certain number of characteristic values into a set of binary attributes. Each characteristic value is assigned a new binary attribute that takes the value 1 if the value occurs in a data record and otherwise has the value 0, see → Example 4.7 and → Section 5.4.1. This method can, for example, transform the attribute *purchasing behavior* with the values *frequent buyer*, *infrequent buyer*, and *non-buyer* in such a way that the attribute *purchasing behavior* is replaced by three binary attributes, namely *frequent buyer*, *infrequent buyer*, and *non-buyer*. In this way, a nominal attribute is converted into several binary-encoded attributes. This form of binary encoding is called *one-hot encoding*.

A modified binary encoding is based on the unary coding, which simply contains as many *ones* as the number we want to represent. Let's look again at the attribute *purchasing behavior* and encode the values according to their order:

$$\textit{frequent buyer}/2 > \textit{infrequent buyer}/1 > \textit{non-buyer}/0$$

In the unary representation, the number 2 is represented by two ones (1 1), the number 1 by a 1, and the number 0 by zero ones. We fill the code consisting only of ones with zeros and obtain a binary code that maintains the order and even has one binary digit less than the previous code:

$$\textit{frequent buyer}/(1\ 1) > \textit{infrequent buyer}/(1\ 0) > \textit{non-buyer}/(0\ 0)$$

When applying binary codes, the performance of data analysis can be impaired by the increasing number of attributes.

Discretization: Conversion of metric data into nominal or ordinal data

The technique of discretization is used to summarize the range of values for numerical attributes into a finite number of subsets. Some methods cannot process numerical attributes and therefore require discretization.

Example 8.5

(Encoding, body height). If we want to use the attribute *body height* when generating a decision tree using ID3, we must convert it into an ordinal attribute.

- *Variant 1:* If there are only a small number of values, we can convert the numerical value into a string.
- *Variant 2:* We form intervals, for example [1.00 - 1.60], and give them names such as *small*. We can do this manually or use an automatic method, such as clustering.

Variant 2 is a form of *binning* that we have already discussed as a discretization technique (p. 207). The binning variants presented there operate *unsupervised*. The intervals are formed either

- by individually defining the interval limits or

- by generating intervals of equal size.

The latter variant can be modified so that each interval contains an equal number of elements. In both variants, the number of intervals to be formed is specified.

Example 8.6

(Binning). We have the measured values {20, 25, 27, 28, 29, 32, 37, 46, 56, 68, 70, 70} and want to sort them into three groups (bins). This can be implemented, for example, as follows:

- *Equal-sized bins (equal frequency)*: We form intervals with an equal number of elements.
 - Bin 1: {20, 25, 27, 28}
 - Bin 2: {29, 32, 37, 46}
 - Bin 3: {56, 68, 70, 70}
- *Replace with mean values*: We replace all values in these intervals with the mean value.
 - Bin 1: {25, 25, 25, 25}
 - Bin 2: {36, 36, 36, 36}
 - Bin 3: {66, 66, 66, 66}
- *Equal-sized bins (equal width)*: Alternatively, we can form intervals of equal length.
 - Bin 1: [20, 36]
 - Bin 2: [37, 53]
 - Bin 3: [54, 70]

For classification data, i.e., data that has been divided into classes, there are a number of *supervised* discretization techniques. These use the additional information provided by the class membership. We consider an attribute A that is to be discretized. For example, we could start by forming intervals for attribute A with the smallest value that A can take. We keep track of the class to which the data record with the smallest value belongs. Now we go through the values of A in ascending order. When the first value appears that belongs to a different class, we introduce an interval boundary and start a new interval. We note the new class again and continue in ascending order.

The method has several weaknesses. The biggest disadvantage is that, under normal circumstances, a large number of intervals are created. It would be better if the number of intervals could be specified in advance. For this reason, methods have been developed which, analogous to Shannon's entropy as discussed in the context of the ID3 algorithm (→ Section 5.2.4) perform interval formation in such a way that the "disorder" with respect to the target attribute is as low as possible.

Clustering methods such as k-means (→ Section 6.2) can also be used for creating intervals. This can be done both unsupervised and supervised. The supervised variant of course requires a class attribute, which is included in the clustering process.

All of these discretization methods are explained in detail in [Cio+10, → Chapter 8].

Normalization and scaling

The terms *normalization* and *scaling* are often used synonymously. In our book, we also do not make a distinction between the two, but this is actually incorrect. *Scaling* is the more general term. Scaling refers to the *transformation* of values to any scale. Normalization is a form of scaling, whereby all values are scaled to a specific value range. Here are some methods for normalization and scaling:

Min-max normalization The data is linearly transformed into a numerical range, for example, the interval $[0, 1]$. The smallest value is scaled to 0, the highest value to 1.

Z-value normalization The normalized value is obtained by dividing the difference between the value and the mean by the standard deviation.

Decimal scaling The data is transformed by shifting the decimal point. This is a linear scaling. It achieves a transformation into a specified interval, for example $[0, 1]$.

Logarithmic scaling The logarithm of the value is calculated with respect to a suitable base B , i.e., $x_{new} = \log_B(x)$

Normalization, or standardization, thus transforms all feature values of an attribute to the values of a continuous, numerical scale.

For min-max normalization to the interval $[0, 1]$, only the minimum value $\min(x_i)$ and maximum value $\max(x_i)$ of the attribute are required. The normalized value is calculated as

$$x_{new} = \frac{x - \min(x_i)}{\max(x_i) - \min(x_i)} \quad 8.1$$

If you do not want to normalize to the interval $[0, 1]$ but rather to an arbitrary interval $[a, b]$, the conversion formula is

$$x_{new} = a + (b - a) \times \frac{x - \min(x_i)}{\max(x_i) - \min(x_i)} \quad 8.2$$

Z-value normalization calculates the statistical mean A and standard deviation δ_A of the attribute values for an attribute A , subtracts the mean from each value, and then divides the result by the standard deviation:

$$x_{new} = \frac{x - \bar{A}}{\sigma_A} \quad 8.3$$

The mean value thus becomes zero. The new values are now on both sides of 0. The relative distances between the values remain unchanged. Z-value normalization is therefore often referred to as *standardization*.

In KNIME, three normalizations are implemented in the *Normalizer* node:

Min-max normalization The minimum and maximum can be freely selected.

Z-score (Gauss) Linear transformation so that the mean is 0 and the standard deviation is 1.

Decimal scaling as described above.

Normalization can be applied if the attribute is metric and if the minimum and maximum of the attribute are given.

As an example of normalization let's consider the encoding of *age*. Here, we will assume that the age values are all between 18 and 67, i.e., the minimum value is 18 years and the maximum value is 67 years. Using min/max normalization on a scale of 0 to 1, an age of 40 would then be encoded as $\frac{40-18}{67-18} = 0.449$. Decimal scaling to the interval $[0, 1]$ causes the values to be divided by 100. The decimal point is therefore shifted two positions to the left: 43 becomes 0.43.

Let's clarify the need for normalization with an example.

Example 8.7

(Normalization). Consider the following table containing data on patients.

No.	Age	Pulse	Blood pressure	Condition
1	27	irregular	117	healthy
2	65	irregular	123	ill
3	33	regular	127	healthy
4	55	irregular	150	ill
5	38	irregular	120	ill
6	36	regular	121	healthy
7	53	regular	140	ill
8	26	regular	107	healthy
9	43	regular	111	healthy
10	60	regular	112	healthy

We want to apply the k-nearest neighbor algorithm (→ Section 5.1) and encode the nominal attribute *pulse* as numerical attribute, since k-nearest neighbor works with numerical attributes.

Target attribute *condition* remains unchanged.

No.	Age	Pulse	Blood pressure	Condition
1	27	0	117	healthy
2	65	0	123	ill
3	33	1	127	healthy
4	55	0	150	ill
5	38	0	120	ill
6	36	1	121	healthy
7	53	1	140	ill
8	26	1	107	healthy
9	43	1	111	healthy
10	60	1	112	healthy

We test k-nearest neighbor ($k = 2$) using the last data record (patient 10). The KNIME workflow is shown in → Figure 8.6.

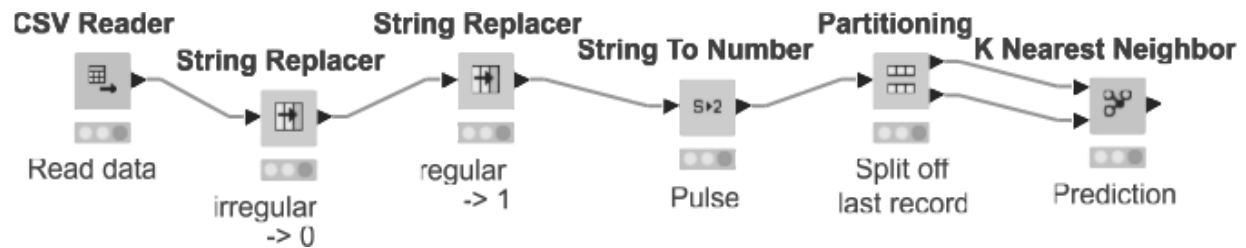


Fig. 8.6 K-nearest neighbor, variant 1

Unfortunately, the algorithm fails here. Patient 10 is predicted to be *ill*, but the patient is *healthy*. Looking at the data, we see that the two closest neighbors are independent of *pulse*. Based on the Euclidean distance, the *pulse* attribute provides a maximum difference of 1. It is therefore clear that the distance measure is dominated by *blood pressure* and *age*. We can avoid this by normalization. Each numerical attribute is normalized to the interval $[0, 1]$ (→ Figure 8.7).

No.	Age	Pulse	Blood pressure	Condition
1	0.0256	0.0	0.2326	healthy
2	1.0	0.0	0.3721	ill
3	0.1795	1.0	0.4651	healthy
4	0.7436	0.0	1.0	ill
5	0.3077	0.0	0.3023	ill
6	0.2564	1.0	0.3256	healthy
7	0.6923	1.0	0.7674	ill
8	0.0	1.0	0.0	healthy
9	0.4359	1.0	0.0930	healthy
10	0.8718	1.0	0.1163	healthy

Now the prediction for data record 10 is correct: *healthy*.

Data normalization requires experimentation. First, we normalize to the interval $[0, 1]$. If we do not obtain satisfactory results, we can normalize attributes to $[0, X]$ instead of $[0, 1]$. The larger X , the greater the influence of

the attribute. The same effect can be achieved with weighted Euclidean distance (see → Section 2.3).

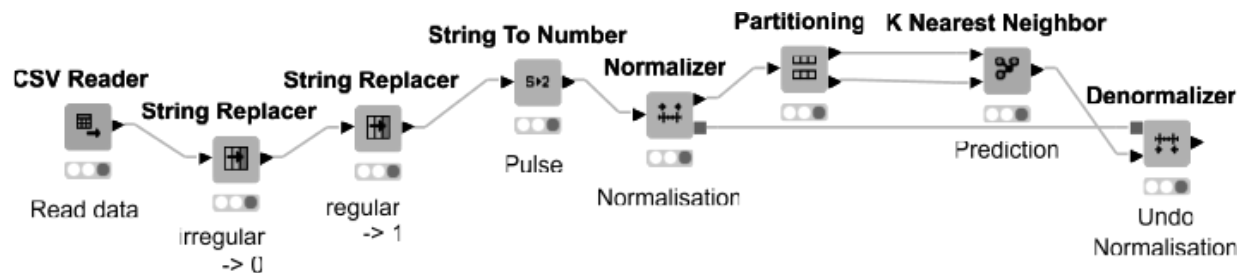


Fig. 8.7 K-nearest neighbor, variant 2

Combining or separating attributes

It may be necessary to *combine different attributes into a new one*, such as day, month, and year into a *date*. Conversely, it may also be necessary to break down the date into its components in order to identify, for example, the beginning of the month or the day of the week. When analyzing customer purchasing behavior, for example, knowing the day of the week can be crucial.

Calculating derived values

By calculating *derived values*, completely new attributes can be added. For example, the attribute *profit* can be calculated as the difference between the attribute values *revenue* and *costs*. Age can be determined as the difference between the *current date* and the *date of birth*.

Data aggregation

Data aggregation can be useful not only from the perspective of data compression (→ Section 8.2.3, p. 210), but also for practical reasons. The data is often available at too fine a level of aggregation. If, for example, you are looking for the population of Berlin and only have the population figures for the individual districts, you can use *aggregation* (in this case, summation) to transfer the data to a higher level of aggregation. Only then is it possible to make meaningful statements about Berlin.

Data smoothing

The main idea behind data smoothing is that each numerical value from the given data set is replaced by idealized values, which are obtained, for example, through regression. The aim of the method is to replace the original set of values with a reduced set, in the hope of finding better solutions. It also reduces noise in the data. Data can be smoothed using the following techniques:

- Binning
- Clustering
- Combined human and machine control
- Regression

We have already discussed this when dealing with noisy data and outliers (p. 207).

Principal component analysis

A frequently used data transformation technique is *principal component analysis* (PCA)¹. Let us consider the two-dimensional data shown in → Figure 8.8, belonging to three classes.

x	y	Class
0	0	class1
2	1.7	class1
1.5	0.5	class1
1	0	class2
2.0	0.55	class2
3	0	class2
0.5	1	class3
1.7	1.9	class3
1.3	1.4	class3

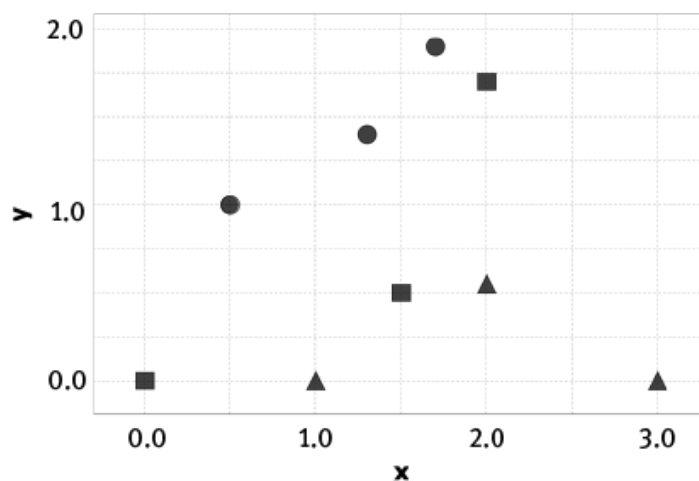


Fig. 8.8 Example data

We cannot expect the C4.5 algorithm (→ Section 5.2.6) to generate a compact decision tree for this data, as this method works axially parallel.

For example, if C4.5 selects x as the root attribute, it separates the values on the x -axis into those that are smaller and those that are larger than a threshold value. This will not result in a simple decision tree for the given data.

However, if we abandon the x and y axes and choose a different axis, it is very easy for C4.5 to generate a compact decision tree. If we rotate the x -axis slightly, as shown by the dashed line in → Figure 8.9, then C4.5 can separate the data according to the classes, as shown by the dotted vertical lines.

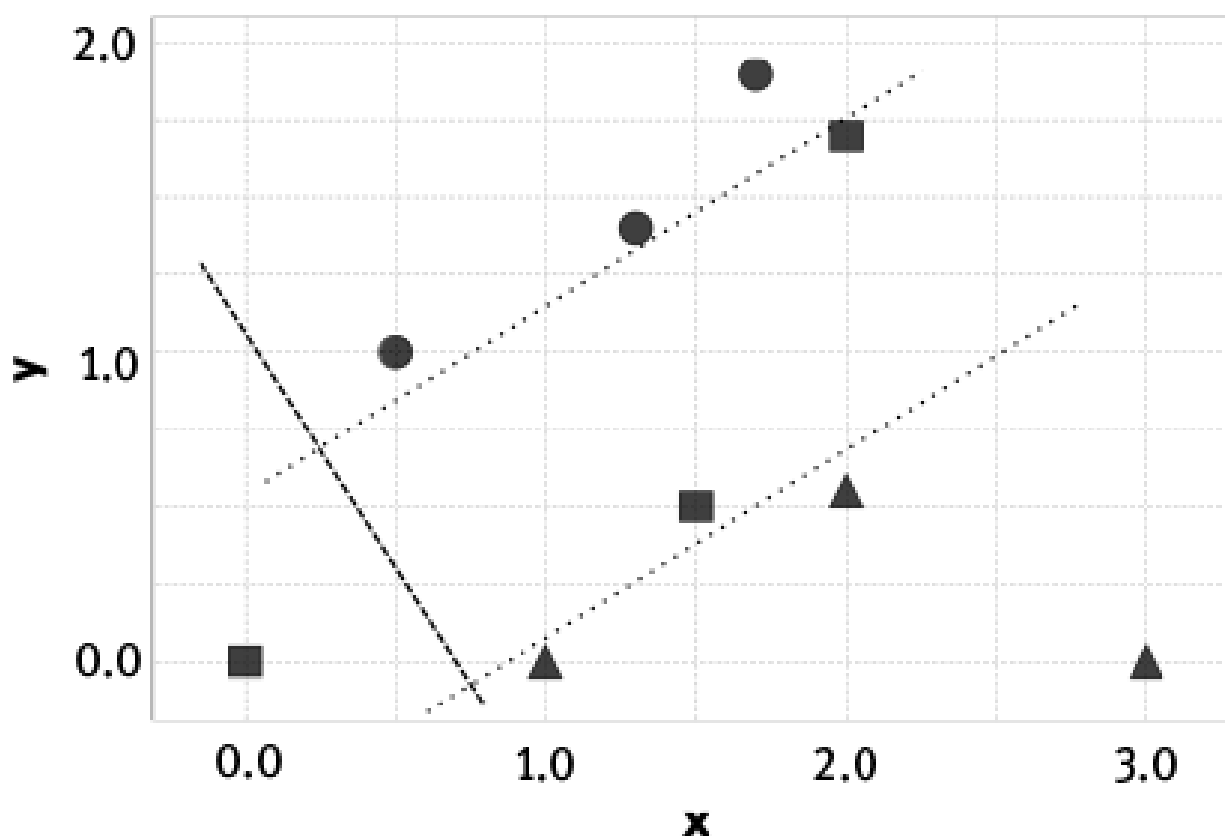


Fig. 8.9 Alternative x -axis

This is the transformation that the *principal component analysis* performs. We will refrain from going into the mathematical details here and simply outline the procedure.

All data records x are converted using a PCA transformation matrix.

$$x_{\text{new}} = \text{PCA} \cdot (x - \bar{x})$$

We use $\bar{x}$ to denote the vector of the average values of the x components.

The matrix PCA can be an $n \times n$ matrix. In this case, the data is only *rotated*. However, it can also be used for dimension reduction, in which case the matrix has the size $m \times n$, where $m < n$.

Which matrix are we looking for? We are looking for the matrix that maximizes the variance in the projection world, because the goal is good separability of the data. This leads to an eigenvalue problem. We obtain m vectors that are precisely the principal axes. These vectors form the transformation matrix.

The PCA transformation is integrated into KNIME. And indeed, the C4.5 in the KNIME variant J48 generates the simple decision tree shown in → Figure 8.10.

The second level is only necessary in this decision tree because J48 always divides an interval into only two subintervals.

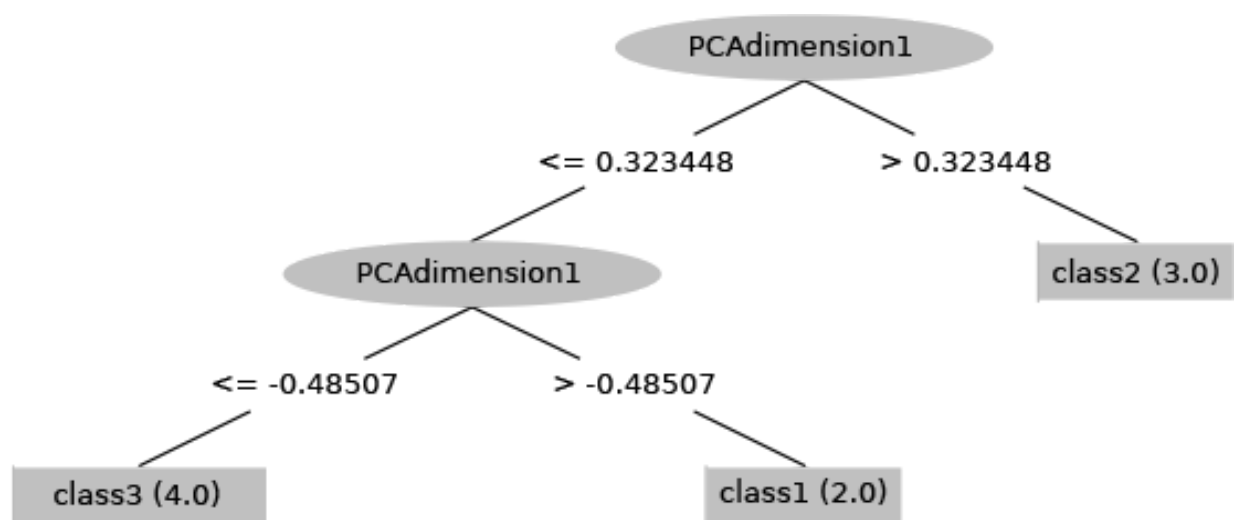


Fig. 8.10 Decision tree after PCA transformation

We do not have to assign the latest data preparation techniques to *data transformation* (step 3 of the Fayyad model, → Figure 1.1). From a data quality perspective, we can also assign these conversion techniques to *data preprocessing* (step 2 of the Fayyad model). It is not always possible to make a clear distinction, but this is not necessary either.

8.3 An example

Let's illustrate a few aspects with an example. Suppose we have already combined our data from different sources into a *single* data table (→ Table 8.2).

Tab. 8.2 Example data for data preprocessing

Pers. no.	Postal code	Gender	Age	Annual salary	Length of service	Position	Educational attainment
1	20095	m	45	32	10	production	apprenticeship
2	20095	f	57	35000	25	administration	Bachelor's degree
3	20095	m	52	40	5	management	Master's degree
4	20095	m	28	27	6	production	apprenticeship
5	20095	male	57	45	25	management	Master's degree
6	20095	fem	26	27	96	production	apprenticeship
7	20095	m	39	39	4	management	Master's degree
8	20095	m	38	32	3	production	apprenticeship
9	20095	m	42	31	15	production	none
10	20095	f	37	30	10	administration	high school diploma
11	20095	m	45	32	8	production	
12	20095	m	37		5		
13	20095	f	35	30	15	administration	high school diploma

What is striking?

- The postal code is the same for all persons (they all live in Hamburg). This attribute will not help us in our analyses, because we need different values for the attributes in order to be able to establish differences or similarities between the objects. We can therefore delete this attribute and thereby reduce the dimension of the problem.
- Gender is specified as both m/f and male/female.
- In data record 2, the annual salary is not specified in thousands.

- Several values are missing in data record 12. We will have to omit this data record.
- In data record 6, the length of service is given as 96. There are two possible interpretations: 96 could be interpreted as 1996, i.e., length of service since 1996. However, it could also refer to the length of service in *months*. We have decided on the second option here, since it is impossible to have been employed since 1996 at the age of 26.
- In data record 11, the educational attainment is missing. We could correct this, for example, by inserting a special value (none) or the most common value that occurs for the specified position (production): apprenticeship. However, we must be aware at this point that by doing so, we are already interpreting something into the data, namely a possible connection between educational attainment and position. Alternatively, we could use a method such as k-nearest neighbor (see → Section 5.1) to find a value for educational attainment based on the nearest neighbor(s).

The modified data is shown in → Table 8.3.

Tab. 8.3 Example data, revised table

Pers. no.	Gender	Age	Annual salary	Length of service	Position	Educational attainment
1	m	45	32	10	production	apprenticeship
2	f	57	35	25	administration	Bachelor's degree
3	m	52	40	5	management	Master's degree
4	m	28	27	6	production	apprenticeship
5	m	57	45	25	management	Master's degree
6	f	26	27	8	production	apprenticeship
7	m	39	39	4	management	Master's degree
8	m	38	32	3	production	apprenticeship
9	m	42	31	15	production	none
10	f	37	30	10	administration	high school diploma
11	m	45	32	8	production	apprenticeship
13	f	35	30	15	administration	high school diploma

Obviously, the personnel number does not contain any information. Information such as department affiliation cannot be identified here. Length of service could be determined from this ID in an ordinal sense, but this is explicitly given anyway. We can therefore delete the personnel number.

Now we need to incorporate the goal of our analysis into the data preparation. Let's start by assuming that we want to cluster the data. Metric values are useful for clustering because they are easier to calculate and therefore produce better distance measures. We can encode the nominal attributes using integer values (→ Table 8.4).

Tab. 8.4 Example data, encoded by numerical values

Gender	Age	Annual salary	Length of service	Position	Educational attainment
0	45	32	10	0	1
1	57	35	25	1	3
0	52	40	5	2	4
0	28	27	6	0	1
0	57	45	25	2	4
1	26	27	8	0	1
0	39	39	4	2	4
0	38	32	3	0	1
0	42	31	15	0	0
1	37	30	10	1	2
0	45	32	8	0	1
1	35	30	15	1	2

Using this table for clustering (KNIME, 3 clusters), we obtain the clusters shown in → Figure 8.11.

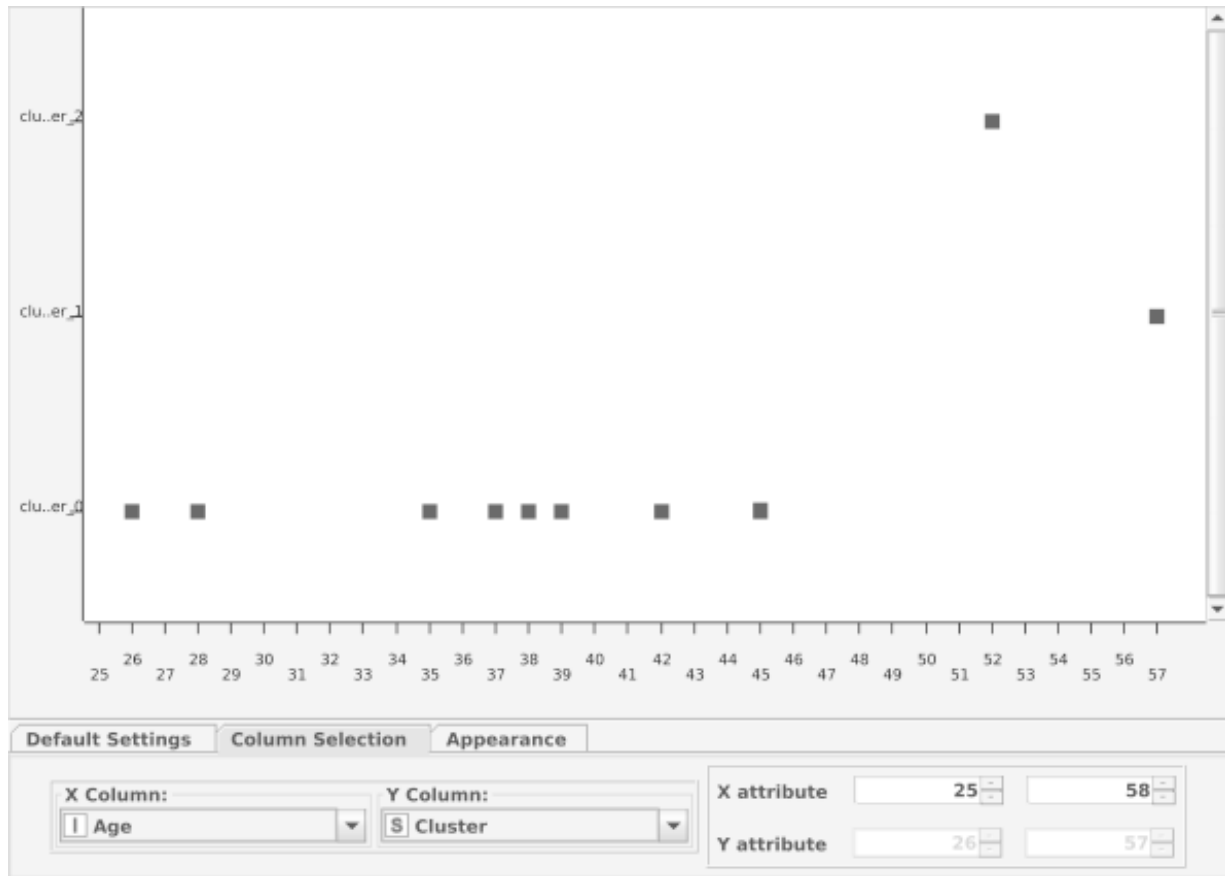


Fig. 8.11 Clustering, experiment 1

It is striking that *age* obviously plays a major role. What is the reason for this? The difference of 31 years between a 26-year-old and a 57-year-old is so significant that the differences in the other attributes play a negligible role. Only the attributes *annual salary* and *length of service* provide significant differences, but these are smaller than those for *age*. The attributes *position* and *educational attainment* provide only minimal differences. How can we eliminate the dominance of the *age* attribute? We can achieve this by normalizing all data to the interval $[0, 1]$, see → Table 8.5.

Tab. 8.5 Example data with normalized numerical values

Gender	Age	Annual salary	Length of service	Position	Educational attainment
0	0.61	0.28	0.32	0	0.25
1	1	0.44	1	0.5	0.75
0	0.84	0.72	0.09	1	1
0	0.06	0	0.14	0	0.25
0	1	1	1	1	1
1	0	0	0.23	0	0.25
0	0.42	0.67	0.05	1	1
0	0.39	0.28	0	0	0.25
0	0.52	0.22	0.55	0	0
1	0.35	0.17	0.32	0.5	0.5
0	0.61	0.28	0.23	0	0.25
1	0.29	0.17	0.55	0.5	0.5

This results in the clusters shown in → Figure 8.12 (again with regard to *age*).

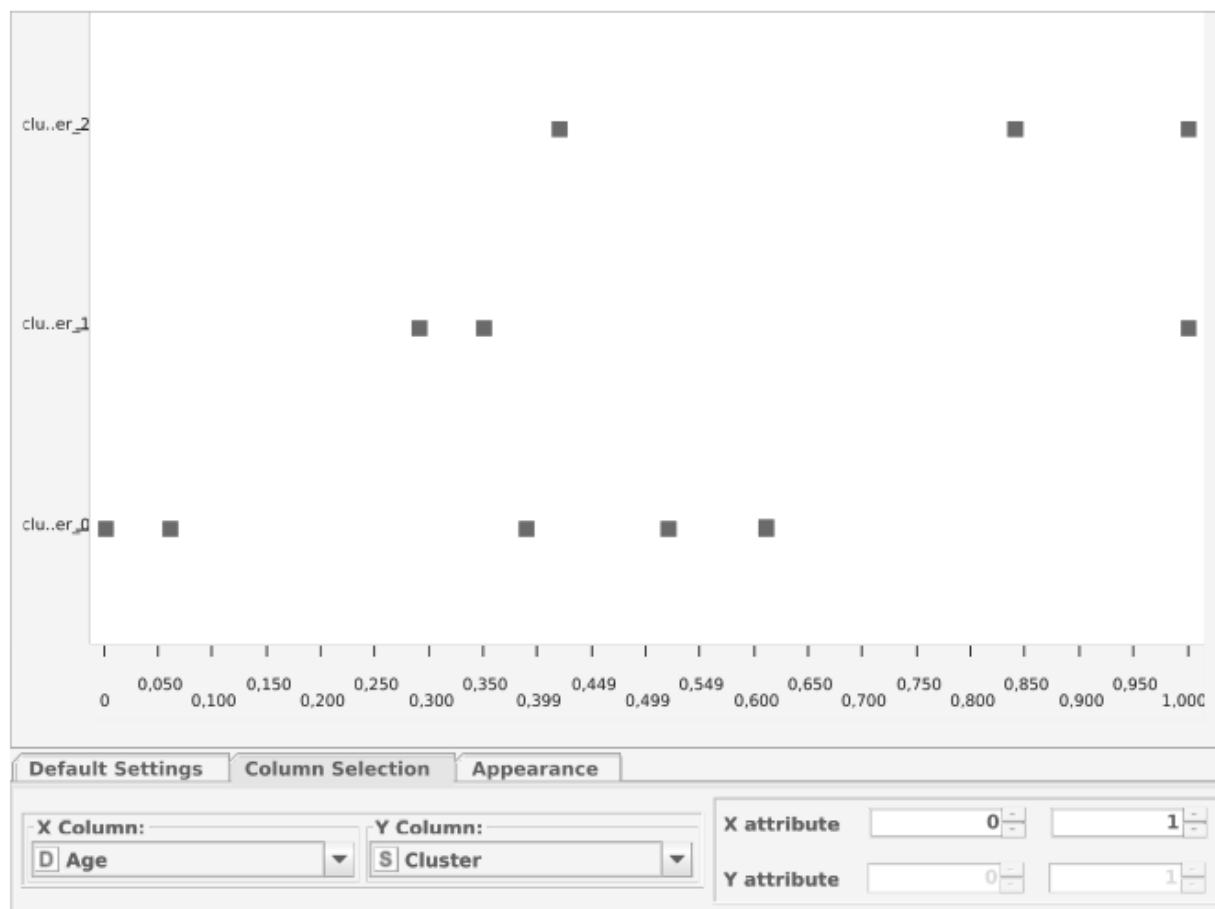


Fig. 8.12 Clustering, experiment 2: age

When looking at the initial data, our assumption is that *educational attainment* is an important attribute. This assumption is confirmed by the new cluster formation, as we can see from the representation of the clusters with regard to *educational attainment* in → Figure 8.13. By normalizing the data, we have obtained a clustering that appears to make sense.

Let's return to the original data. Now we want to generate a decision tree for the target attribute *educational attainment*, using ID3. We don't need metric attributes, but rather nominal or ordinal attributes. We therefore have to convert the metric attributes into ordinal attributes, see → Table 8.6. To do this, we perform clustering on all values of *one* attribute, thus obtaining two intervals, which are named accordingly.

The ID3 algorithm can now work with this data. The decision tree shown in → Figure 8.14 is generated with the target attribute *educational attainment*.

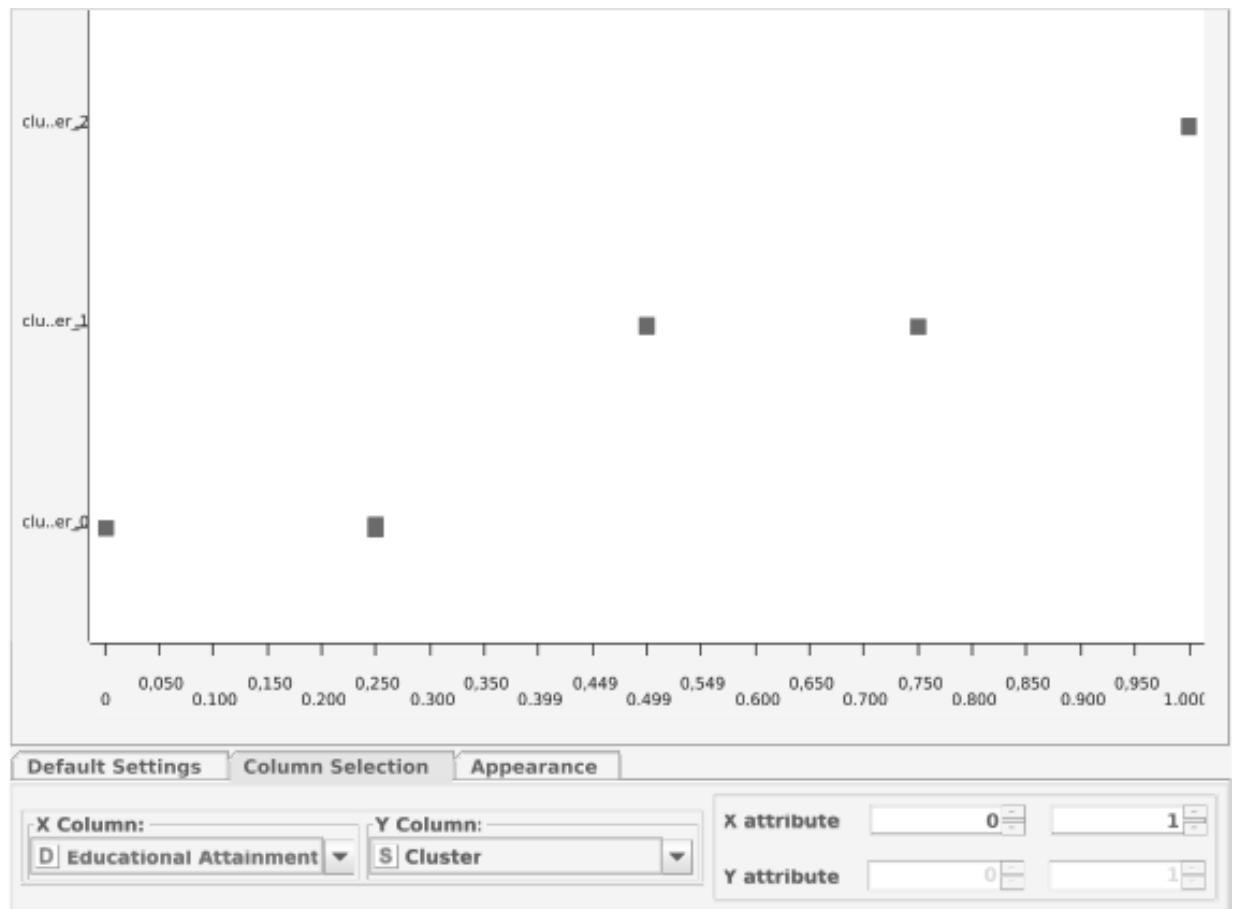


Fig. 8.13 Clustering, experiment 3: educational attainment

Tab. 8.6 Example data with nominal values

Gender	Age	Annual salary	Length of service	Position	Educational attainment
m	old	low	short	production	apprenticeship
f	old	high	long	administration	Bachelor's degree
m	old	high	short	management	Master's degree
m	young	low	short	production	apprenticeship
m	old	high	long	management	Master's degree
f	young	low	short	production	apprenticeship
m	young	high	short	management	Master's degree
m	young	low	short	production	apprenticeship
m	old	low	long	production	none
f	young	low	short	administration	high school diploma
m	old	low	short	production	apprenticeship
f	young	low	long	administration	high school diploma

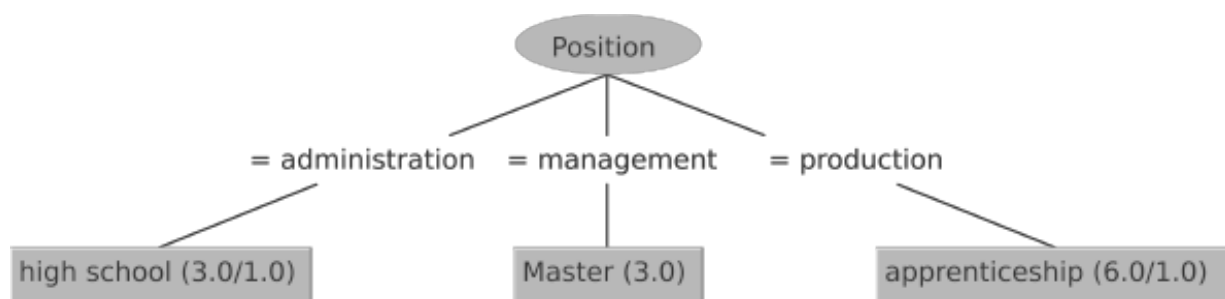


Fig. 8.14 Decision tree generated by ID3

Task 8.1

(Data preprocessing for contact lens data). What data preprocessing steps are necessary for the contact lens data (→ Appendix A.4) in order to cluster the data?

Task 8.2

(Data preprocessing for Iris data). What data preprocessing steps are necessary for the Iris data (→ Appendix A.1) if classification is to be performed using ID3, kNN, or naive Bayes?

Task 8.3

(Data preprocessing). Consider the following table containing information about buyers of dishwashing detergent.

No.	Gender	Age	Income	Brand
1	female	29	low	Magic View
2	male	21	high	Super Fairy
3	female	18	medium	Super Fairy
4	male	49	low	Magic View
5	female	44	high	Clear & Spotless
6	male	21	high	Clear & Spotless
7	male	43	high	Clear & Spotless
8	male	24	low	Magic View
9	female	19	medium	Super Fairy
10	male	45	high	Clear & Spotless
11	female	34	low	Magic View
12	male	25	medium	Super Fairy

This table is to be used to perform a classification by k-nearest neighbor. The target attribute is *brand*. Does the data need to be preprocessed?

Task 8.4

(Data preprocessing, other examples). Go through the other examples in the book and examine the data preprocessing. What preprocessing is necessary to be able to apply the methods discussed in → chapters 5, → 6 and → 7?

9 Evaluation

Negative expectations yield negative results. Positive expectations yield negative results.

Nonreciprocal Laws of expectations

Knowledge is hidden in data sets in various ways. Each data mining method only searches for specific correlations. It is possible that information contained in the data remains undiscovered because the method cannot see or represent these relationships. It therefore makes sense, for example, to work with both neural networks and decision trees in a classification task. In order to assess the quality of the models obtained, we need comparability, which is achieved through quantitative evaluation.

We will now look at the final step in a data mining process according to Fayyad's model, the assessment and evaluation of the results obtained, see → Figure 1.1 (p. 3). Criteria that should be applied to a data mining result have already been defined in → Section 1.3:

1. Validity
2. Novelty
3. Comprehensibility
4. Usefulness

There are a number of other requirements, but these are subsumed by the above criteria. For example, *conspicuousness*, i.e., the deviation of the result from standard results, falls under *novelty*. The literature also mentions the criterion of *general validity*, which falls under *validity*.

To evaluate these criteria, we consider several measures for the interestingness. First, it should be noted that a method or a developed model can only be evaluated if the expected result is available for comparison or if the quality of the solution can be assessed in some other way. In supervised learning, this is possible because we know the result. In decision tree learning, we can count how often the decision tree correctly predicts class assignment and how often it predicts it incorrectly.

When evaluating clustering methods, it is more complicated to decide whether a method or cluster formation is good or not. In real-world tasks, the target clusters are not known, yet it is still possible to compare different clusters that are generated using different methods but the same data set.

9.1 Principle of minimum description length

Before we discuss some evaluation measures, let's start with a basic principle:

If several methods are evaluated as equally good in terms of result, preference is given to the method or model that presents the relationships being sought in the simplest and most easily comprehensible way.

This principle is called the *minimum description length* (MDL) principle. The description length is defined as

- Storage space for describing a theory plus
- Storage space for describing the errors (exceptions) of the theory.

This concept goes back to William of Ockham, born around 1285 in Ockham (England), who postulated that the simpler a model is, the better it is. This is known as *Occam's razor* or the principle of parsimony.

This principle also plays a role in the generation of decision trees: compact, i.e., small trees are desirable.

9.2 Measures for the interestingness of association rules

The result of an association analysis (see → Section 4.4) is often a large number of rules. The problem-related interpretation and evaluation of the rules must be carried out by the user. This can be a problem, as the number of rules is often difficult to manage and there is no general procedure with uniform evaluation criteria. This problem is also referred to as the *post-mining rule analysis problem*.

The goal of a meaningful interpretation is to adequately reduce the sheer volume of information:

- It is possible to sort the rules according to certain significant characteristics, such as *support* or *confidence*. This makes it easier to assess the significance of a rule.
- The set of rules can be filtered by accepting only rules that meet certain criteria. For example, rules whose support is less than 20 % are removed. Taking the market basket example, you could search for rules that contain the terms *coffee* and *milk*.
- If the rules are not very complex, graphical representations can be used to identify correlations between certain attributes.

If no explicit analysis goal is pursued, these measures quickly end up in a trial-and-error approach that is not very effective. It therefore makes sense to use

interestingness measures to evaluate rules. These allow the results of an association analysis to be evaluated on a mathematical basis.

9.2.1 Support

Support was already introduced in → Section 4.4. Support is a measure for the frequency with which the combination of precondition and postcondition of a rule $M \rightarrow N$ occurs in the data records. It takes real values between 0 and 1 and indicates the proportion of the total amount covered by a rule.

Unfortunately, it is not possible to make a general statement about the support values a rule must have in order to be worth considering further. The support merely indicates which facts occur frequently and which occur rather rarely. Nevertheless, rare items can also contain interesting correlations.

The support of a rule $M \rightarrow N$ is defined by

$$\text{supp}(M \rightarrow N) = P(M \cup N) \quad 9.1$$

We calculate the frequency P of transactions in which both M and N occur. The support can be illustrated using a Venn diagram, see → Figure 9.1: it is the area of the shaded region $M \cap N$, divided by the area of the entire set.

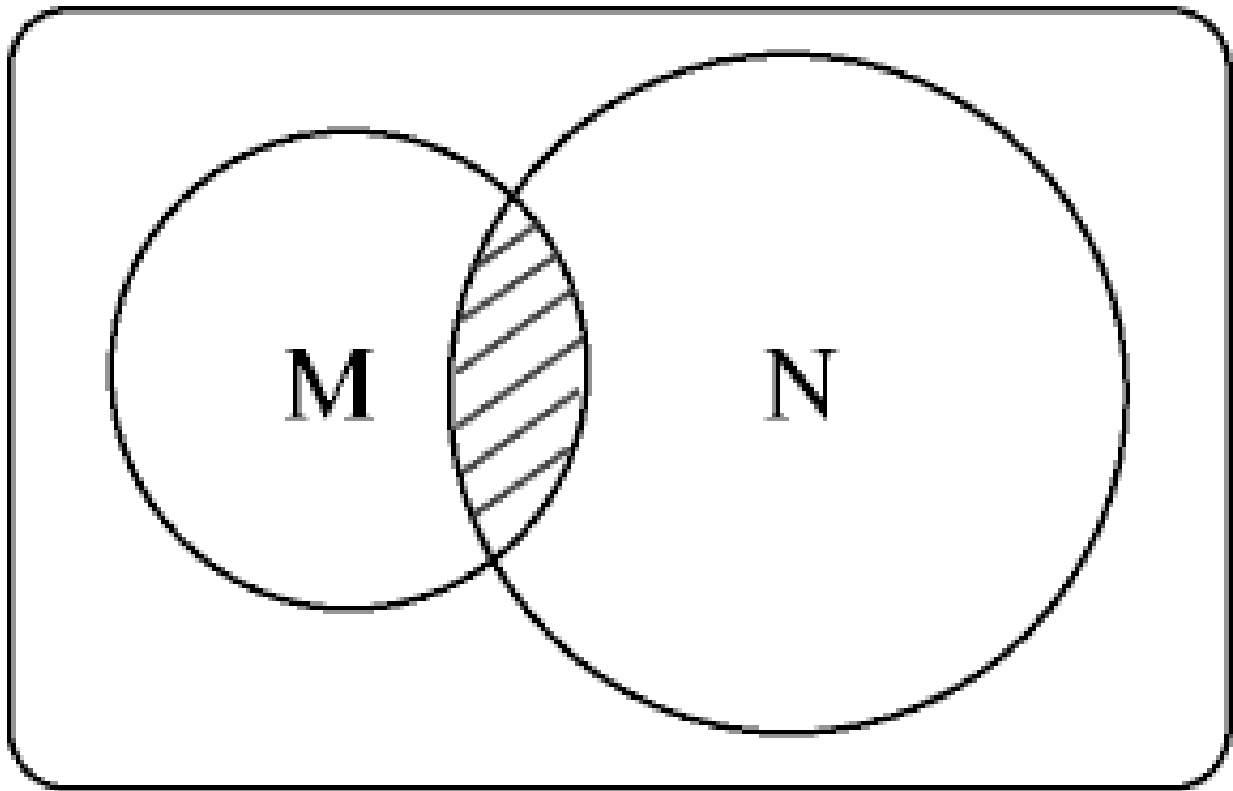


Fig. 9.1 Support of the association rule $M \rightarrow N$

9.2.2 Confidence

Confidence represents the strength of the correlation of M and N . It is defined as the conditional probability:

$$\text{conf}(M \rightarrow N) = P(N|M) = \frac{\text{supp}(M \rightarrow N)}{\text{supp}(M)} \quad 9.2$$

Similar to support, confidence is also measured in values between 0 and 1.

Confidence can also be illustrated using a Venn diagram (→ Figure 9.2.): it is the ratio of the area shaded with solid lines (cases in which both M and N apply) to the area M shaded with dashed lines.

One disadvantage of confidence is that even rules with a low statistical correlation can have a high confidence. This is because the calculation of confidence does not take into account the probability of N .

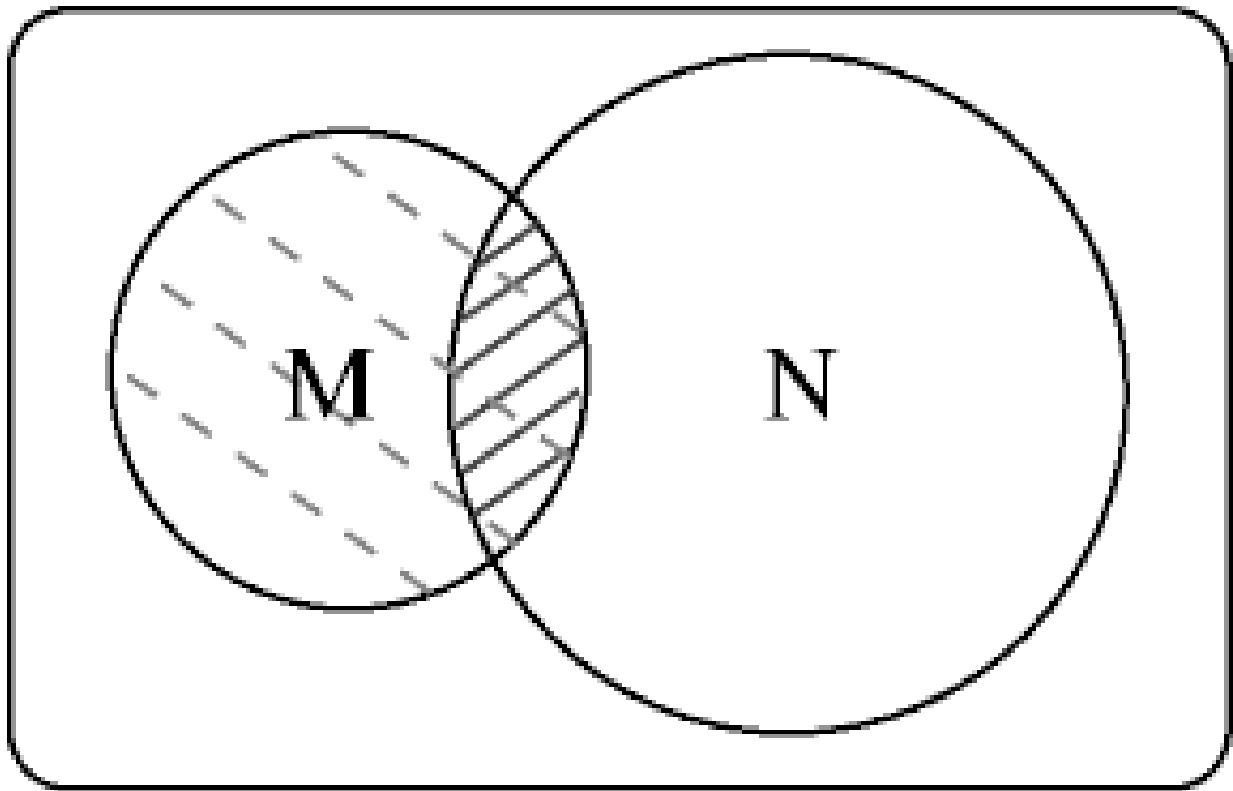


Fig. 9.2 Confidence of the association rule $M \rightarrow N$

Example 9.1

(Confidence). Using the example of the shopping basket, it can be argued that bread and milk are purchased very frequently in absolute terms. It therefore makes little sense to relate them to other products. Let us assume that bread is *always* purchased. Then the rule

IF truffles THEN bread

has a confidence of 1.0. We can replace truffles with other products: the confidence remains 1.0. This statistical correlation is obviously questionable.

It therefore makes sense to look at other measures for the interestingness in addition to support and confidence.

9.2.3 Completeness

The shopping basket problem described in → Example 9.1 can be addressed by introducing another measure of interestingness, namely *completeness*, which is defined as follows:

$$\text{completeness}(M \rightarrow N) = P(M \cup N | N) = \frac{\text{supp}(M \rightarrow N)}{\text{supp}(N)}$$

9.3

Completeness corresponds to the ratio of the area $M \cap N$, i.e., all cases in which both M and N apply, to the area N (→ Figure 9.3). Thus, it describes the proportion of N covered by the rule.

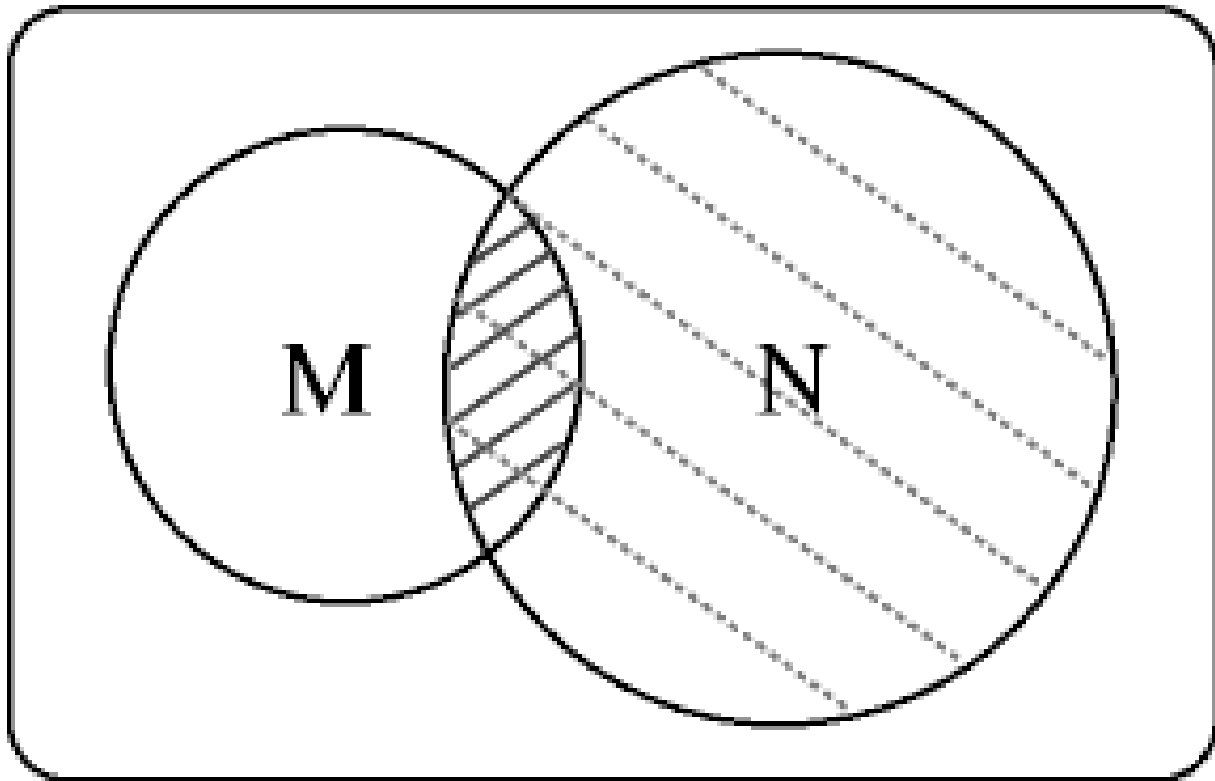


Fig. 9.3 Completeness of the association rule $M \rightarrow N$

Example 9.2

(Interestingness measures for association rules). Given the following frequency values:

- $P(M) = 0.50$
- $P(N) = 0.40$
- $P(M \cup N) = 0.35$

For the measures of interestingness, we then obtain the following values:

$$\text{supp}(M \rightarrow N) = P(M \cup N) = 35\%$$

$$\text{conf}(M \rightarrow N) = P(N|M) = \frac{\text{supp}(M \rightarrow N)}{\text{supp}(M)} = \frac{0.35}{0.50} = 70\%$$

$$\text{completeness}(M \rightarrow N) = P(M \cup N|N) = \frac{\text{supp}(M \rightarrow N)}{\text{supp}(N)} = \frac{0.35}{0.40} = \frac{7}{8} = 87.5\%$$

The confidence is not high, but this is offset by the relatively high completeness, making the rule an interesting candidate after all.

9.2.4 Gain function

It sometimes happens that a rule that narrowly fails to meet the support requirement has a very high confidence level. In this situation, it is annoying that we have defined a rigid limit based on support. However, it is not possible to dispense with support entirely, as this would result in extremely rare rules (with low support) being accepted.

This can be avoided with the gain function, which is a mix of support and confidence:

$$\text{gain}(M \rightarrow N) = \text{supp}(M \rightarrow N) - \theta \cdot \text{supp}(M) \quad 9.4$$

The parameter θ can take values between 0 and 1, which means that the value of the gain function is always between $-\theta$ and $1 - \theta$. A value of $\theta = 0$ is not useful, because then gain and support are identical.

A gain function value of 0 means that in every $1/\theta$ th case in which M is contained, N also occurs. Values above or below 0 symbolize a stronger or weaker correlation, respectively.

Note 9.1

(Gain). Where does the *gain* come from? We require that a rule $M \rightarrow N$, which has a high confidence but does not occur very often (the support of M is low), is less interesting than a rule that has the same confidence but a higher support (of M). The lower the support of M , the higher the confidence of the rule must be:

$$\text{conf}(M \rightarrow N) \geq \frac{\min_{\text{conf}}}{\text{supp}(M)} + \theta$$

High confidence can therefore compensate for low support (and vice versa). We allow θ values between 0 and 1. Since confidence is defined as

$$\text{conf}(M \rightarrow N) = \frac{\text{supp}(M \rightarrow N)}{\text{supp}(M)}$$

we can substitute this into the above inequality and multiply the inequality by $\text{supp}(M)$. We obtain

$$\text{supp}(M \rightarrow N) \geq \text{min}_{\text{conf}} + \text{supp}(M) \cdot \theta$$

and therefore

$$\text{supp}(M \rightarrow N) - \theta \cdot \text{supp}(M) \geq \text{min}_{\text{conf}}$$

The left side of the last inequality is the gain introduced in Equation 9.4.

$$\text{gain}(M \rightarrow N) = \text{supp}(M \rightarrow N) - \theta \cdot \text{supp}(M)$$

9.2.5 *p-s-Funktion*

Gregory Piatetsky-Shapiro → [PS91] addressed the shortcomings of some measures for the interestingness of association rules. He proposed that measures of interestingness for association rules (rule interestingness, *RI*) must meet the following criteria:

1. $RI(M \rightarrow N) = 0$ applies if and only if $\text{supp}(M \rightarrow N) = P(M) \cdot P(N)$. The measure of interestingness should be 0 if M and N are statistically independent.
2. RI increases monotonically with $\text{supp}(M \rightarrow N)$.
3. RI decreases monotonically with $P(M)$ and $P(N)$.

Criterion 1 deals with the situation where M and N are statistically independent of each other. What does that mean? How many correct predictions for N would we expect by chance? Without any restrictions, the probability of randomly selecting a purchase that satisfies N is simply $P(N)$, i.e., the relative frequency of N . Suppose, we now additionally require that M is also satisfied. If M and N are independent, then $P(M) \cdot P(N)$ should result. Criterion 1 therefore requires that $RI = 0$ if the support of the rule $M \rightarrow N$ behaves exactly as it would if item sets satisfying M and N were selected at random, assuming that M and N are independent.

Criterion 2 requires that RI increases with the support of the rule. More precisely: RI increases with the support of the rule if $\text{supp}(M)$ and $\text{supp}(N)$ remain constant.

Requirement 3 states that RI decreases monotonically with increasing relative frequency of M . The same must be true for N . Here, it must be required that the other values remain constant.

Confidence only satisfies condition 2. In particular, criterion 1 is often violated, and even independent sets M and N are usually evaluated with high confidence. Confidence also does not satisfy criterion 3, since it does not depend on $P(N)$.

The p -s function satisfies the above restrictions. For $\theta = \text{supp}(N)$, it is identical to the gain function:

$$ps(M \rightarrow N) = \text{supp}(M \rightarrow N) - \text{supp}(M) \cdot \text{supp}(N) \quad 9.5$$

The p -s function assumes that the support of a rule should be higher than that of statistical independence. Positive values reflect a positive correlation, and negative values reflect a negative correlation.

9.2.6 Lift

Dividing the p -s function by $\text{supp}(M) \cdot \text{supp}(N)$ and ignoring the constant subtrahend 1 yields the *lift*:

$$\text{lift}(M \rightarrow N) = \frac{\text{supp}(M \rightarrow N)}{\text{supp}(M) \cdot \text{supp}(N)} = \frac{\text{conf}(M \rightarrow N)}{\text{supp}(N)} \quad 9.6$$

The lift indicates the extent to which the distribution of an element in a subset differs from that in the total set. Values greater than 1 correspond to a positive correlation. Thus, a lift of 1 corresponds to a correlation of zero. For example, while a lift of 4 for the rule $M \rightarrow N$, for example, means that N occurs four times more frequently within this rule than in the total set.

Example 9.3(Lift).

Bread is purchased in 40 % and water in 30 % of cases. Both products together are purchased in 20 % of cases. What do we get for the rule water $\rightarrow$ bread ?

Support 20%

Confidence $\frac{0.2}{0.3} = 0.66$

Completeness $\frac{0.2}{0.4} = 0.5$

Lift $\frac{0.66}{0.4} = 1.66$

The confidence is not overwhelming, but at least it is above 50 %. The completeness tells us that the rule is not particularly interesting, as only half

of bread purchases are covered by the rule. From the lift, we can conclude that bread and water are purchased together more often than could be explained by pure chance; i.e., there is a relationship.

9.2.7 Categorization of interestingness measures

The measures of interestingness for a rule $A \rightarrow B$ pursue different objectives, which are presented here in the form of questions:

Confidence How certain is the rule? If precondition A is fulfilled, what is the probability that B can be concluded?

Support How universal is the rule? How often do A and B occur together in the entire test set?

Lift Is inferring B from A better than chance or not? In other words, does predicting B using this rule lead to better results than pure chance?

Completeness How many B cases are actually covered by the rule? The higher the completeness, the better the rule is from the perspective of predicting B as completely as possible using A .

Task 9.1

(Evaluation of association rules). Calculate the support, confidence, and completeness as well as the gain, p -s value, and lift for the rule $M \rightarrow N$. We have 50,000 data records in which M occurs 900 times and N occurs 1200 times. M and N occur together 800 times.

9.3 Quality measures and error costs

Now that we have looked at measures of interestingness for association rules, let's turn to the question of how to measure the quality of a decision tree or k -nearest neighbor classification. One simple option is to calculate the error of a prediction: the expected (real) value is compared with the predicted value.

9.3.1 Error rate

Error rate in classifications In classification problems, the *error rate* describes the relative proportion of incorrectly classified examples:

$$\text{error rate} = \frac{\text{incorrect class assignments}}{\text{all class assignments}}$$

Error rate for numerical prediction In numerical prediction, this absolute statement is supplemented by the difference between the predicted and expected

results:

$$\text{error rate} = \frac{\sum_i (\text{real value}_i - \text{predicted value}_i)^2}{\sum_i \text{real value}_i^2}$$

There are other measures for evaluating numerical predictions, see [Wit+25].

Success rate In order to convey a subjectively better impression, the success rate is often given instead of the error rate:

$$\text{success rate} = 1 - \text{error rate}$$

9.3.2 Additional quality measures of classifiers

There are quality measures specifically for classifiers that extend the concept of error rate, some of which we will now examine. To do this, we will consider a classifier that predicts *good* customers. Four different prediction cases can be distinguished.

TP	true positive	Number of correctly recognized positive cases: <i>A good customer is recognized as good.</i>
TN	true negative	Number of correctly recognized negative cases: <i>A poor customer is recognized as poor.</i>
FP	false positive	Number of cases incorrectly recognized as positive: <i>A poor customer is recognized as a good.</i>
FN	false negative	Number of cases incorrectly recognized as negative: <i>A good customer is recognized as poor.</i>

Based on this, derived characteristic values are defined [Run25, → Section 8.1]:

Correct classifications	$T=TP+TN$	all correct predictions
Incorrect classifications	$F=FP+FN$	all incorrect predictions
Relevance	$R=TP+FN$	number of <i>good</i> customers
Irrelevance	$I=FP+TN$	number of <i>poor</i> customers
Positivity	$P=TP+FP$	number of customers classified as <i>good</i>
Negativity	$N=TN+FN$	number of customers classified as <i>poor</i>
Accuracy rate	T/n	proportion of <i>correctly</i> classified customers
Incorrectness rate	F/n	proportion of <i>incorrectly</i> classified customers
True positive rate	$TPR=TP/R$	How often was a <i>good</i> customer classified as such? (sensitivity, hit rate, recall)
True negative rate	$TNR=TN/I$	How often was a <i>poor</i> customer classified as such?
False positive rate	$FPR=FP/I$	How often was a <i>poor</i> customer classified as <i>good</i> ?
False negative rate	$FNR=FN/R$	How often was a <i>good</i> customer classified as <i>poor</i> ?
Positive predictive value	TP/P	How often is a customer predicted to be <i>good</i> a <i>good</i> customer? (accuracy, precision)
Negative predictive value	TN/N	How often is a customer predicted to be <i>poor</i> a <i>poor</i> customer?
Negative false classification rate	FN/N	How often is a customer predicted to be <i>poor</i> a <i>good</i> customer?
Positive false classification rate	FP/P	How often is a customer predicted to be <i>good</i> a <i>poor</i> customer?

A single measure of quality is usually not sufficient. For example, a 90 % accuracy rate can be very misleading. Let's consider an example where 90 % of the data records belong to class 1 and the remaining 10 % to class 2. If we simply predict class 1 for all data records, we immediately achieve 90 % accuracy, but we are far from having a good classifier. This is exactly the situation with the data discussed in → Chapter 10.

Similarly, we can construct examples for the quality metrics TPR, TNR, FPR, FNR (and also the others) in which the respective metrics are not helpful because they refer to only one aspect (TP, TN, FP, or FN) of the classification.

Example 9.4

(Metrics for classifiers). Suppose, a classification has yielded these results:

- Of 1,000 customers, 300 belong to the *poor* class and 700 to the *good* class.
- Our method correctly classified 290 of the 300 poor customers; of the 700 good customers, 650 were correctly classified.

We calculate some of the above metrics as examples.

- TP=650, TN=290, FP=10, FN=50
- Correct classifications: T=TP+TN = 940
- Incorrect classifications: F=FP+FN=60
- Relevance: R=TP+FN = 700
- Irrelevance I=FP+TN=300
- Recall: TPR=TP/R=650/700
- Positivity: P=TP+FP=650+10=660
- Precision: TP/P=650/660

In order to reduce the potential bias of the metrics discussed above, they are also combined. For example, *precision* and *recall* can be combined to form the so-called

F-measure:

$$2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad 9.7$$

The F-measure is the harmonic mean of *precision* and *recall*. The value of the F-measure should be close to 1. This is the case when *precision* and *recall* are not too far from 1.

For Example 9.4, we obtain the F-measure:

$$F - \text{measure} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = 2 \cdot \frac{\frac{650}{660} \cdot \frac{650}{700}}{\frac{650}{660} + \frac{650}{700}} = 0.956$$

Example 9.5

(F-measure). Let us consider a classifier for 100 data points with the following values:

- TP=10, TN=10, FP=0, FN=80
- Correct classifications: T=TP+TN = 20
- Incorrect classifications: F=FP+FN=80
- Relevance: R=TP+FN = 90
- Irrelevance: I=FP+TN=10

Our classifier therefore only classifies 20 % of the examples correctly. *Precision* focuses only on the positively classified cases, which is P=TP+FP=10, and thus yields 1.

- Precision: $\frac{TP}{P} = \frac{10}{10} = 1$

The *recall* recognizes the weakness of the classifier:

- Recall: $\text{TPR} = \frac{\text{TP}}{\text{R}} = \frac{10}{90} = 0.1111$

The *F-measure* now combines these two values to:

- F-measure: $2 \cdot \frac{0.1111 \cdot 1}{0.1111 + 1} = 0.2$

The aim of a prediction is to achieve the highest possible accuracy. Now that we have discussed metrics for quality, we will discuss how to improve quality.

→ Chapter 5 deals with classification methods. Many of these methods have degrees of freedom: we can vary threshold values, learning factors, or other parameters of the respective method. For example, assuming two classes in the following, the naive Bayes method (→ Section 5.3) can be modified in such a way that the probabilities are first calculated and then, instead of simply predicting the class with the higher probability, the decision is made using a threshold value.

→ Chapter 10 contains data that only includes two classes: cancelers (*yes*) and non-cancelers (*no*). Using naive Bayes, we predict the class *yes* if the probability exceeds a given threshold value. For example, if a customer has a probability of 0.4 of belonging to the class *yes*, then they are classified as non-canceling with a threshold value of 0.5, but as canceling (*yes*) with a threshold value of 0.3. If this threshold value is varied, different hit rates result. In a *receiver operating characteristic* (ROC) curve, the true positive rate (TPR) and the false positive rate (FPR) are plotted against each other. → Figure 9.4 shows this for the data from → Chapter 10 (Workflow 10.6).

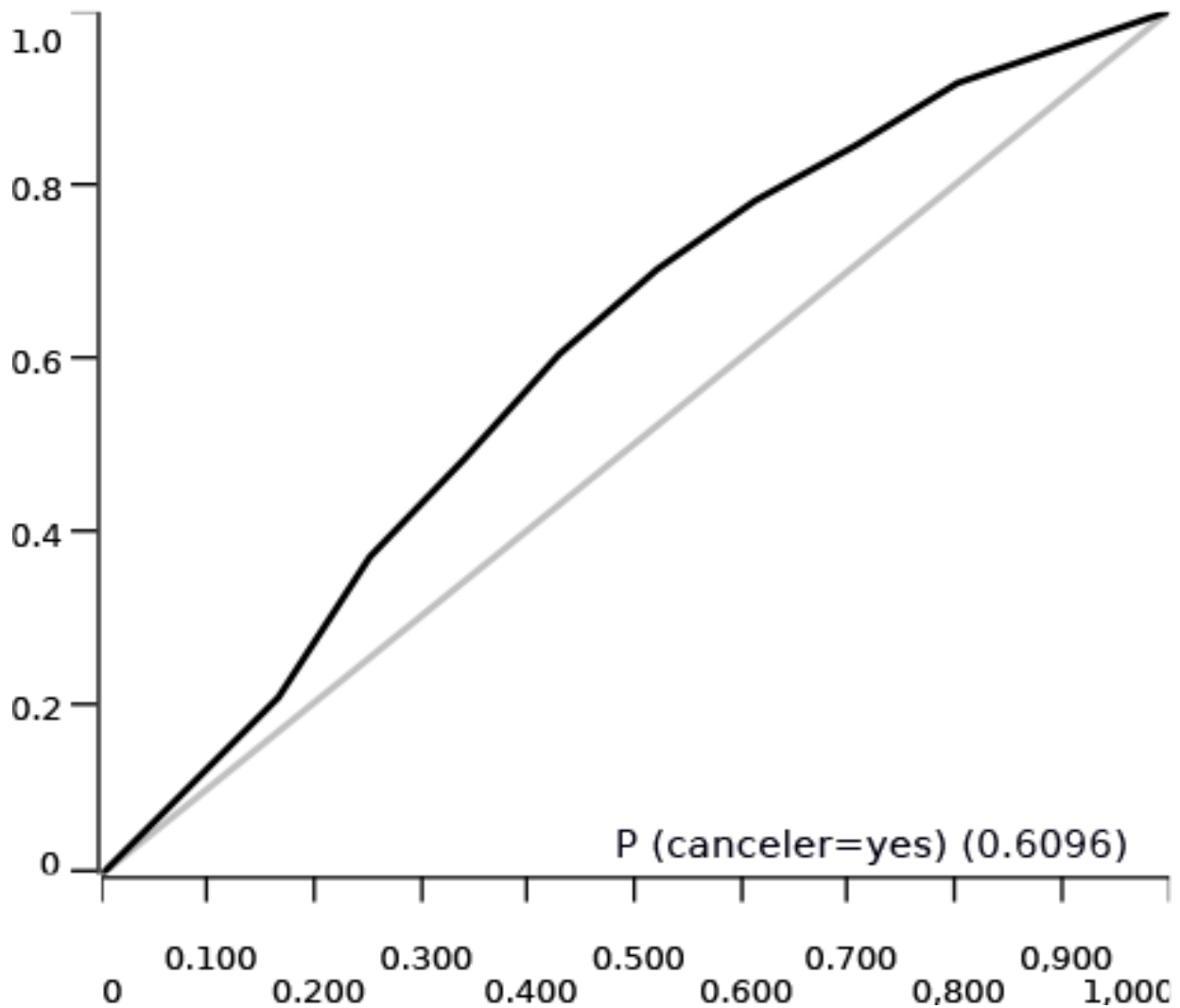


Fig. 9.4 ROC curve

The false positive rate (FPR) is shown on the x-axis, i.e., the percentage of non-cancelers who were incorrectly classified as cancelers. The true positive rate (TPR) is shown on the y-axis, i.e., the percentage of cancelers who were correctly classified. The procedure is now applied with different threshold values, starting with a high value. The threshold value is gradually reduced so that both TPR (the correct yes predictions) and FPR (the incorrect yes predictions) increase. The resulting FPR and TPR values are entered into the diagram. The data points are connected to form a curve.

Obviously, it is optimal if the curve jumps directly to the value 1. This would be the perfect procedure. The diagonal line results when making random predictions. Although the ROC shown in → Figure 9.4 is far from an optimal curve, it is better than a purely random approach. An optimal threshold value for the yes prediction

can be read from this curve by finding the point closest to the ideal point (0, 1) and taking the corresponding threshold value.

This approach is not limited to naive Bayes, but can be applied to other methods as long as these methods provide probabilities for class assignment.

The ROC curves have another advantage: we can now even compare different methods with each other. If the comparison is not obvious, we calculate the area *under* the ROC curve: *Area Under Curve* (AUC). The random (diagonal) curve has an area of 0.5. The ideal curve, which jumps directly to the TPR value of 1, has an area of 1. The larger the AUC value, the better the method.

Alternatively, a PR diagram can be used, which compares *precision* and *recall*.

9.3.3 Error costs

Until now, we have assumed that all errors are equally important for our result. However, this assumption does not apply to many practical applications. From a business perspective, decisions made on the basis of a classification can have different effects on a company's profits or losses. Taking this aspect into account, the so-called *error costs* can also be calculated for a procedure. If, for example, a bank customer is granted a loan based on the positive classification by a decision tree and does not repay it, the bank incurs additional costs, possibly even amounting to the full loan amount. However, if someone who would have repaid the loan is denied credit, the bank only incurs interest losses. This means that not all errors are equal. The cost of an error in deciding to grant a loan can be higher than deciding against it. If different types of errors occur unevenly with their specific costs, what is sought is not the model with the minimum error rate, but the one with the economically optimal error rate.

For this, a cost matrix or bonus matrix is set up, which awards plus or minus points depending on whether the prediction is correct or incorrect.

Prediction		
	0	1
0	10	-20
1	-30	20

If a data record of type 0 is predicted with 1, this costs us 20, while a correct prediction of 0 earns us 10. The goal is now no longer to maximize the percentage of correctly predicted data records, but rather to maximize profit.

Example 9.6

(Data Mining Cup 2002). → Chapter 10 deals with the task of the Data Mining Cup 2002. There, the following cost matrix is given:

Customer wants to cancel	yes	no
Customer is classified as canceler.	€ 43.80	€ 66.30
Customer is classified as non-canceler.	€ 0.00	€ 72.00

If we recognize that a customer is likely to cancel, we make them an offer. On average, we can then expect to generate revenue of € 43.80. If such a customer is not identified, they will cancel and no further revenue can be expected. If we classify a non-canceler as a canceler and make them a better offer, they will gratefully accept it and pay less than before. We then no longer earn € 72, but only € 66.30.

The percentage of correct predictions plays a minor role here. What matters much more is the profit achieved.

9.4 Test sets

All data mining methods for prediction are based on the fact that they “remember” the examples presented to them in some way, even if not explicitly every single one. Thus, every method should correctly process sample data that is presented to it frequently enough. What is more interesting is the question of how the method behaves when presented with new data. It therefore makes sense to keep a stock of examples in addition to the training data, which is used exclusively for testing purposes.

The only data we have available is the sample data. We know the correct prediction or classification for this data. We divide this set into percentages: on one subset, we learn, and on the other, we check whether the learned model works well. In this section, we present methods that divide a set of instances available for training purposes into suitable training and test sets.

Again, we start from the basic assumption formulated in → Section 2.6 that new data records behave in the same way as the existing training data. Without this assumption, data mining does not make sense. One consequence of this assumption is that any partitioning of the given examples must be done in such a way that the training and test sets are *similar* to each other.

In order to be able to compare different data mining methods, ideally the same partitioning of the instance set should be used for all methods. In this way, all methods are trained with the same data and also tested with the same data, ensuring that the error rates are comparable.

Holdout

In *holdout*, a subset of the available set is “set aside” for later testing. The remaining instance set is used for training. In effect, the instance set is partitioned into two subsets, the training set and the test set, i.e.,

$$\text{training set} \cup \text{test set} = \text{instance set}$$

Stratification

With completely random holdout, it may happen that there are patterns in the test data for which no similar patterns exist in the training data. In this case, the method cannot be expected to deliver good predictions on new data for which there are no similar examples in the training data. Our basic assumption, that new data records behave in the same way as the existing training data, requires us to have similar examples in the training data.

Stratification tries to divide the data in such a way that the frequency distribution h of the classes K is the same in both the training and test sets as it is in the entire instance set, i.e., the following should apply:

$$\forall k \in K : h_k^{\text{Instance set}} = h_k^{\text{Training set}} = h_k^{\text{Test set}}$$

The stratification option is available in KNIME during *partitioning* and should always be selected.

→ Figure 9.5 shows a poor training set and the decision tree generated using J48 for the weather example. The training set contains only one example for the case where there is no play. We cannot expect to find a classifier that provides good predictions for *play*.

Outlook	Temperature	Humidity	Windy	Play
sunny	hot	high	false	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
overcast	cool	normal	true	yes
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes

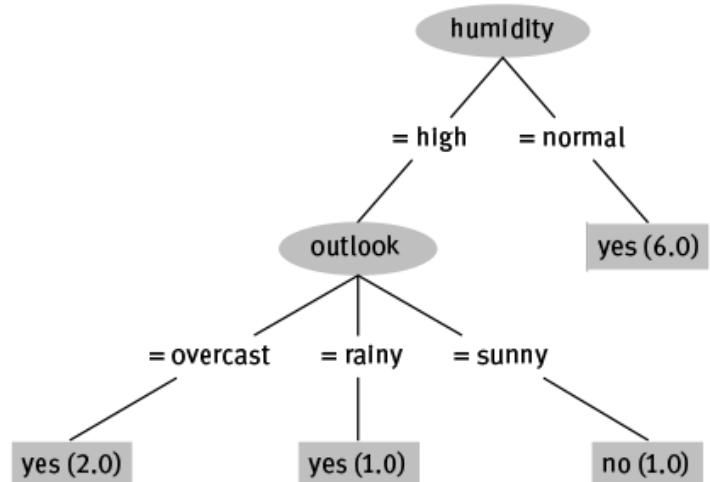


Fig. 9.5 Poor training set and decision tree for the weather example

This effect doesn't just happen with really small amounts of data, but can also occur if the training and test sets are split up in a bad way. The C4.5 algorithm only counts relative frequencies, so there's no difference between a bad split with a small set and a bad split with a big set.

Cross-validation

In cross-validation, the available data set is divided into V equal subsets. One of these subsets is then retained as a test set, while the others are used to train the method. The error rate is then determined on the retained test set. This is repeated with each other subset as the test set. The error rate of the method is calculated as the mean of the respective error rates:

$$\text{error rate} = \frac{\sum_{i=1}^v \text{error rate}_i}{V}$$

If the validation is repeated several times, the mean value of the error rates of the individual validations provides a more accurate error rate for the procedure. A modification of cross-validation is *bootstrapping*, where overlapping samples are allowed. This approach has already been discussed in → Section 5.6.

Leave-one-out

The leave-one-out method is the same as cross-validation. However, in this case, the instance set of size N is partitioned into exactly N subsets, which leads to

$$\text{error rate} = \frac{\sum_{i=1}^{|\text{set of instances}|} \text{error rate}_i}{|\text{set of instances}|}$$

Cross-validation always learns on $N - 1$ elements and tests the classifier on the remaining example.

For some methods, it makes sense to generate a *third set*. Iterative methods such as *boosting* (→ Section 5.6.2) modify the training set by including patterns from the test set that were incorrectly classified in the first run in the training set in the second run. This results in the training and test sets being mixed. Therefore, when using this approach, it is advisable to first separate a subset of the total data set as a *validation set*, which is not used during training. The validation set is only used to test the finished model.

9.5 Quality of clusters

The quality of clustering is far more difficult to evaluate than that of a classifier. The question is:

When is one cluster formation better than another?

We have already addressed this question in → Sections 6.2 and → 6.3 in connection with the k-means and k-medoid algorithms. There are two different approaches.

Approach 1

A first requirement is that the individual clusters should be as compact as possible. The sum of the deviations (or their squares) of the objects in a cluster from the cluster representative is a measure of the compactness of a cluster. The sum of the deviations across all clusters should be minimal. We initially assume that we have specified a *fixed* number k of clusters. The quality of a clustering C can now be measured by

$$G_1 = \sum_{i=1}^k \sum_{x \in C_i} \text{dist}(x, m_i)^2 \quad 9.8$$

Here, m_i is the representative of cluster C_i . The smaller the G_1 value is, the higher the quality of the clustering. Therefore, the reciprocal value is taken as a measure of quality:

$$\text{quality}_1 = \frac{1}{\sum_{i=1}^k \sum_{x \in C_i} \text{dist}(x, m_i)^2} \quad 9.9$$

This approach is discussed in connection with the k-means and k-medoid (→ Sections 6.2 and → 6.3).

Approach 2

Alternatively, it can be required that the clusters be as far apart from each other as possible. The squares of the distances between all cluster representatives are then summed:

$$\text{quality}_2 = \sum_{1 \leq i \leq j \leq k} \text{dist}(m_j, m_i)^2 \quad 9.10$$

Both approaches can be combined by multiplying the quality measures:

$$\text{quality} = \text{quality}_1 \cdot \text{quality}_2 \quad 9.11$$

The maximum distance between two points in a cluster can also be used instead of G_1 . Similarly, you can take the distance between each point and its *nearest* neighbor (in the same cluster) and calculate the maximum value across all points (*minimum distance, single linkage*). These concepts are presented in → Section 6.5 on agglomerative clustering.

So far, we have assumed a *fixed* number of clusters k . Does anything change if k can vary? We are looking for clustering method in which the number of clusters is not predetermined.

Based on the quality metrics defined above, the result is a clustering in which all objects form their own separate cluster. The quality according to quality metric 1 is then maximum, since the denominator is 0.¹ It cannot be better. Quality metric 2 is also maximum.

One way to deal with varying k is to carry out experiments with different k values and then compare the quality of the resulting clusterings with each other.

A second possibility arises from hierarchical clustering (→ Section 6.1.2). In both agglomerative and divisive clustering, certain criteria can be used to find the optimal cluster formation (see → Section 6.5).

A third option is the silhouette coefficient → [Rou87]. This requires the average distance $\text{dist}_1(x)$ of an object x to the other objects in the *same* cluster, as well as the average distance $\text{dist}_2(x)$ that x has to the objects in the *nearest* neighboring cluster. The silhouette coefficient of x is defined by

$$s(x) = \frac{\text{dist}_2(x) - \text{dist}_1(x)}{\max(\text{dist}_1(x), \text{dist}_2(x))}$$

The silhouette coefficient of a cluster is then calculated as the arithmetic mean of the silhouette coefficients of all objects in the cluster. If a cluster consists of only one element, the silhouette coefficient is set to 0.

The silhouette coefficient can take values between -1 and 1. A value close to 1 indicates good separation, while -1 indicates poor separation. Good clusters yield coefficients that are as close to 1 as possible, but at least well above 0.

For the clustering shown in → Figure 9.6, the silhouette coefficient for the element marked ° tends toward -1, since $\text{dist}_2(^{\circ})$, the distance to the nearest cluster, is relatively small, while $\text{dist}_1(^{\circ})$, the distance to the elements of its own cluster, is relatively large.

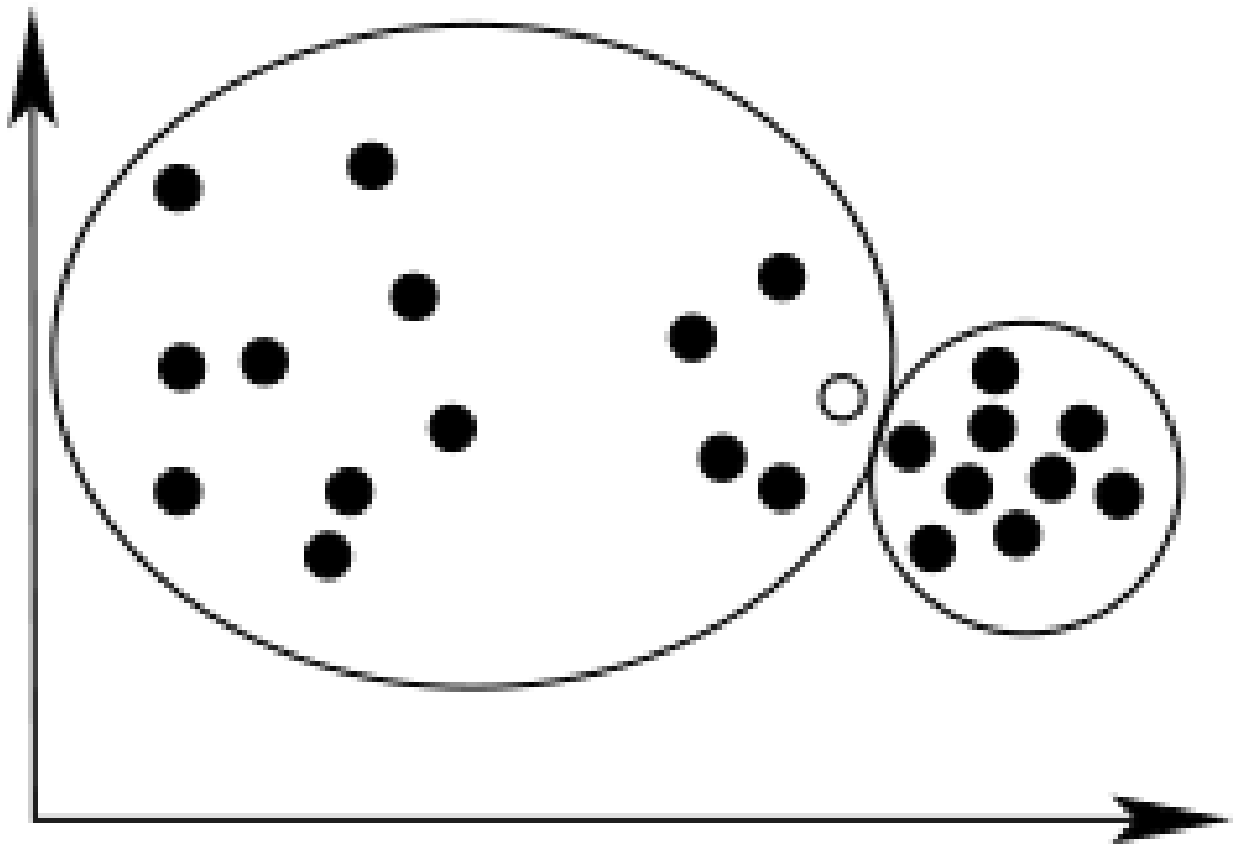


Fig. 9.6 Clustering, variant 1

If the clustering is changed to an obviously better cluster formation, the result is a coefficient that is significantly above zero (→ Figure 9.7).

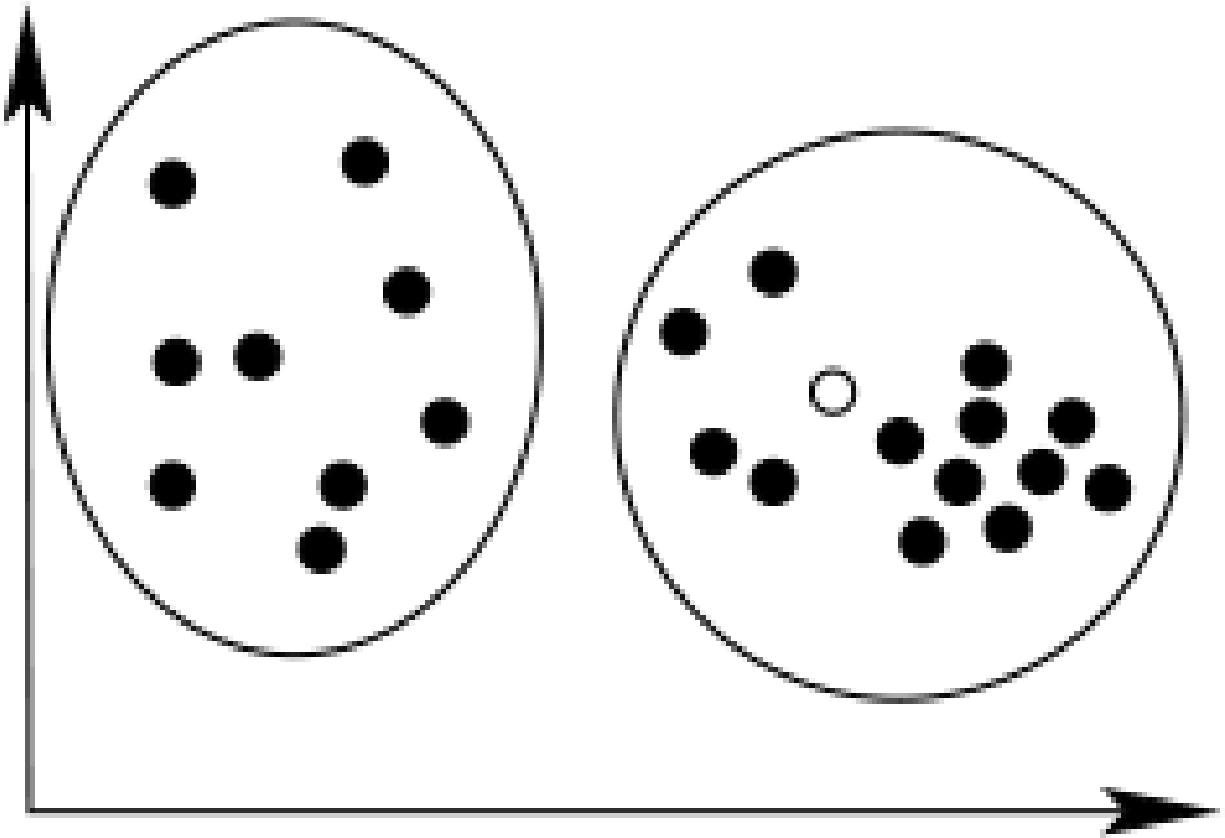


Fig. 9.7 Clustering, variant 2

9.6 Visualization

The goal of a data mining project is to gain knowledge. Be it clusters, a decision trees, or association rules, the models and solutions developed must be illustrated in some way. On the one hand, this serves to improve our own understanding of the problem and the data. In addition, a comprehensible representation of the model or the results increases confidence in the solution developed. On the other hand, we also need to present our results in a comprehensible and traceable form in order to convince our customers, colleagues, or superiors that the results found are correct and valuable.

We humans are not particularly good at comparing columns of numbers. Recognizing patterns, however, is easy for us. Let's look at the Iris data set (see → Appendix A.1). The Iris data set consists of 50 observations, each of which includes four attributes of the flowers (*sepal length*, *sepal width*, *petal length*, and *petal width*, all in cm). Each plant is assigned to an Iris species (*Iris setosa*, *Iris versicolor*, or *Iris virginica*).

```
5.1 , 3.5 , 1.4 , 0.2 , Iris-setosa
4.9 , 3.0 , 1.4 , 0.2 , Iris-setosa
4.7 , 3.2 , 1.3 , 0.2 , Iris-setosa
4.6 , 3.1 , 1.5 , 0.2 , Iris-setosa
.....
```

Humans can glean little or nothing from this data. However, if we visualize the data (→ Figure 9.8), we immediately get a first impression and can identify some correlations.

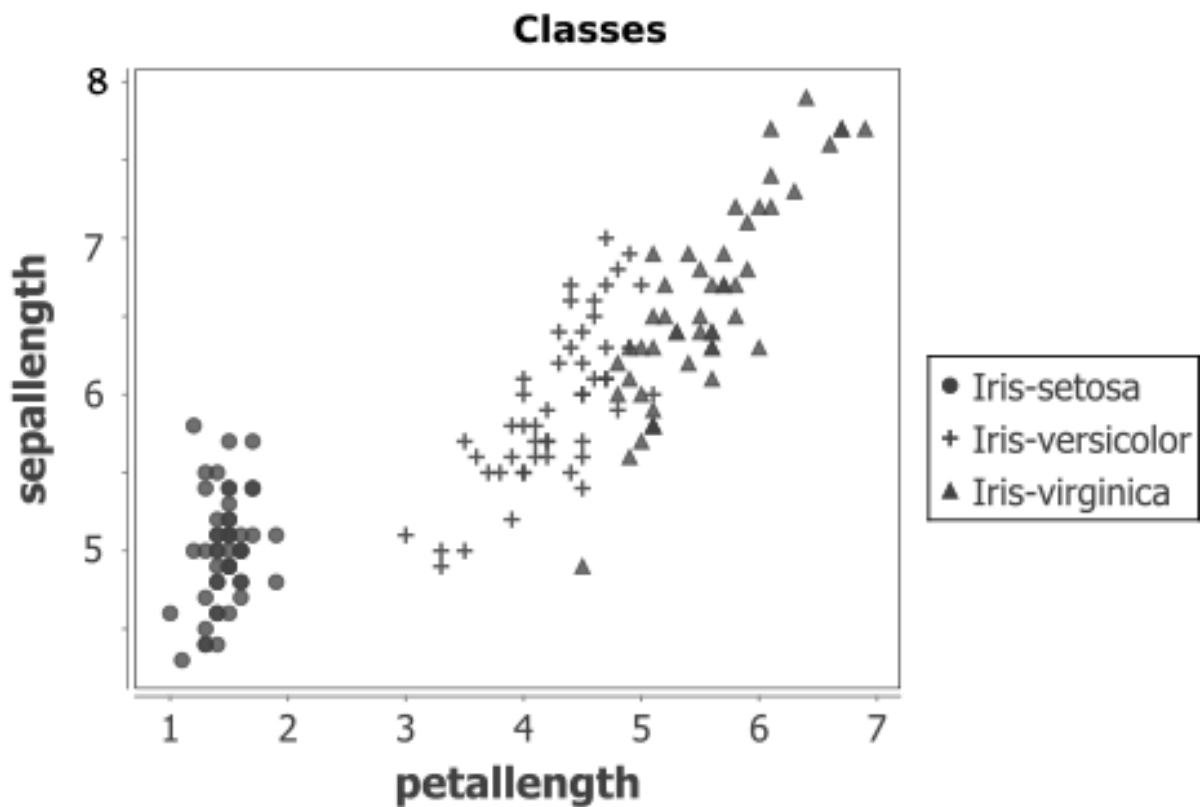


Fig. 9.8 Visualization of Iris data in KNIME

We cannot provide a comprehensive introduction to data visualization here, but will simply illustrate a few possibilities using examples. Naturally, we use some forms of visualization in this book, for example in the representation of *decision trees* or *neural networks*. A decision tree is never stored in its graphical form, such as the Weka representation in Figure → 1.20 (p. 30). However, this form of presentation is well suited for clarity and documentation purposes.

Even rules can be considered a form of visualizations in the broadest sense, even if they are more likely to be regarded as particularly easy-to-read textual representations. Here are a few examples of visualizations.

A simple form of visualization is the *line chart*. It displays two-dimensional information. → Figure 9.9 shows the functions² $f(x) = \cos(2 \cdot \pi \cdot x)$ and $g(x) = \cos(3 \cdot \pi \cdot x)$.

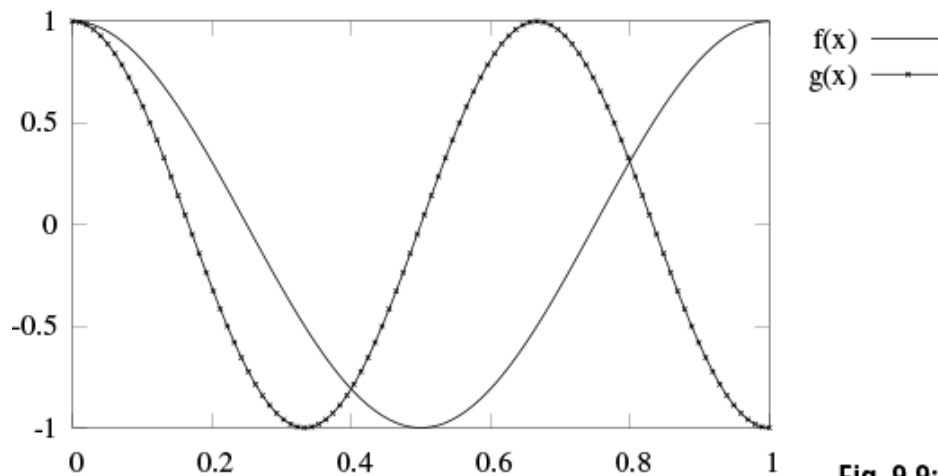


Fig. 9.9: Visualization, line chart 1

Fig. 9.9 Visualization, line chart 1

Here we can already see that colors or patterns, in this case different line types for the two functions, add another dimension.

A line chart can always be used when there is, or is suspected to be, a correlation between two dimensions. → Figure 9.10 shows the values for *sepal length* and *petal length*. Even if this graph is not entirely convincing, it does reinforce the assumption that there may be a correlation between these two attributes.

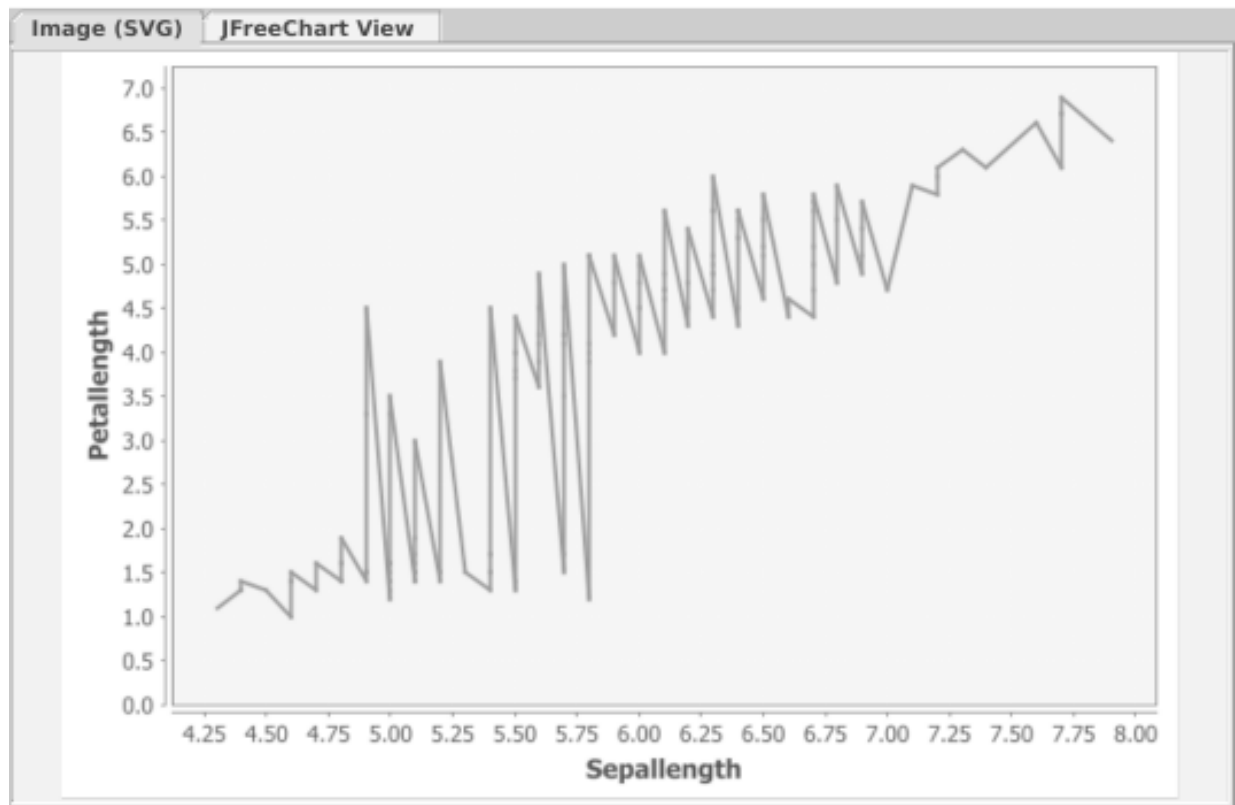


Fig. 9.10 Visualization, line chart 2

Such a chart can be useful in the preparatory phase. For example, we have derived a hypothesis from → Figure 9.10. Visualization can therefore not only help us understand the data, but also formulate plausible hypotheses. Line charts are also very well suited for illustrating the improvement in the quality of a data mining model in our experiments.

A line chart can be used to display values for multiple attributes. → Figure 9.11 shows the data records from the Iris example, sorted by *petal length* and *sepal length*. The x-axis shows the number of the data record, while the y-axis shows the respective value for each of the four attributes.

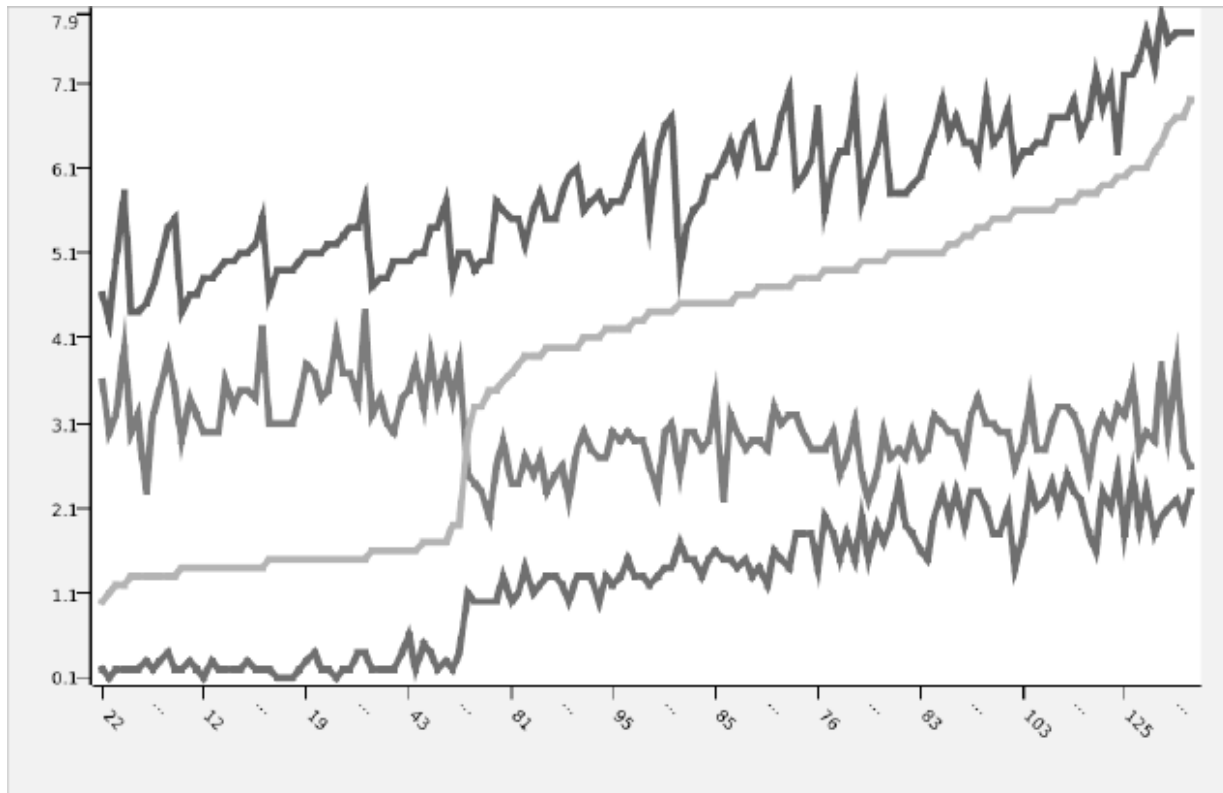


Fig. 9.11 Visualization, line chart 3

We can see from both figures that a line chart is only of limited use in this case, as the values are discrete, the Iris values are rounded to one decimal place. Consequently, interpolation, as performed by the line chart, does not make sense. A scatter plot is more useful and accurate in this case (see → Figure 9.18). However, the line chart emphasizes the possible connection between the attributes more strongly.

For another type of chart, the *bar chart*, we refer to the soybean data set (see → Appendix A.2). The aim is to predict diseases in soybeans. A bar chart illustrates the frequencies of attribute values. → Figure 9.12 shows the frequency distribution of the target attribute, in this case diseases.

This information can be useful, for example, to find out whether a few diseases occur very often or whether there are only a few cases of certain diseases. In particular, too few cases with a certain value of the target attribute can lead to incorrect classifications. If only two cases of a particular disease are given, the use of k-nearest neighbor with a high k value is questionable.

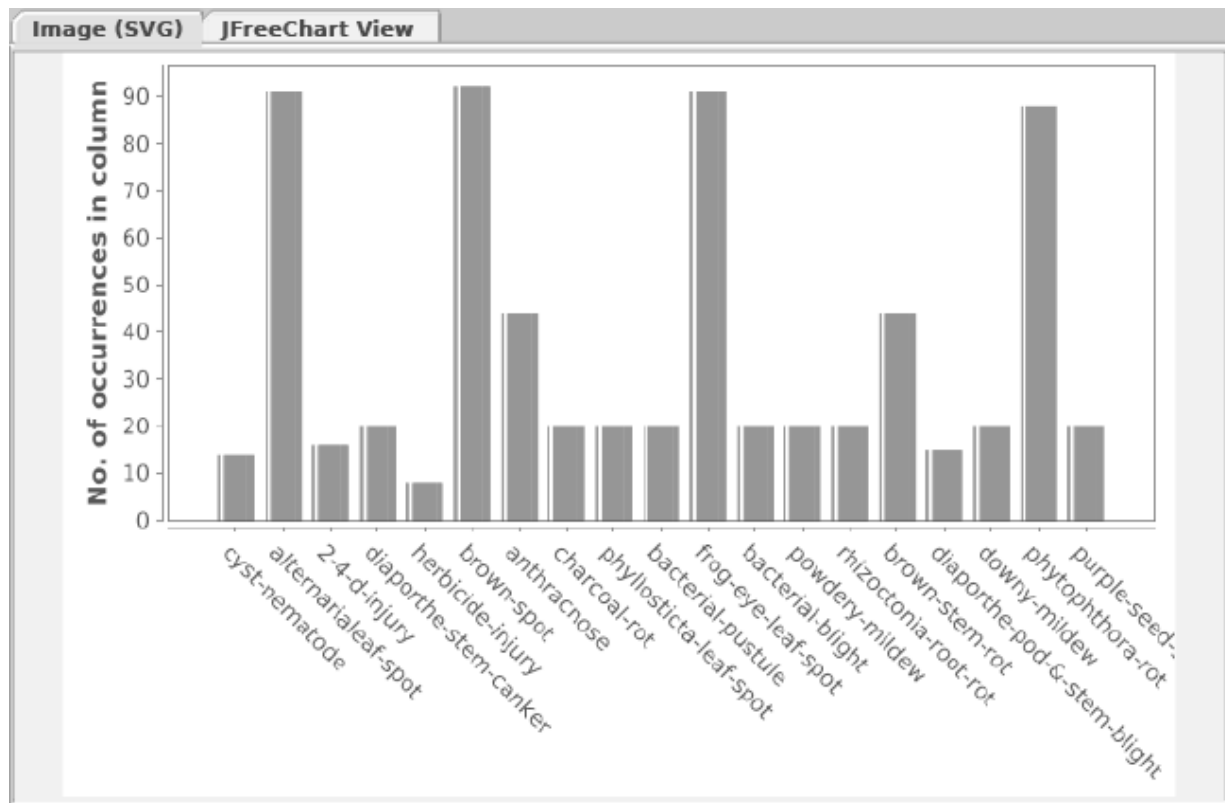


Fig. 9.12 Visualization, bar chart 1

Information about multiple attributes can also be displayed in a bar chart.

→ Figure 9.13 shows the frequency distributions for multiple attributes. This representation only makes sense if the attributes have the same range of values. The selected attributes *plant growth*, *leaves*, *stem*, and *seed* all have the same range of values, consisting of the values *abnorm* and *norm* as well the character ? (for missing value).

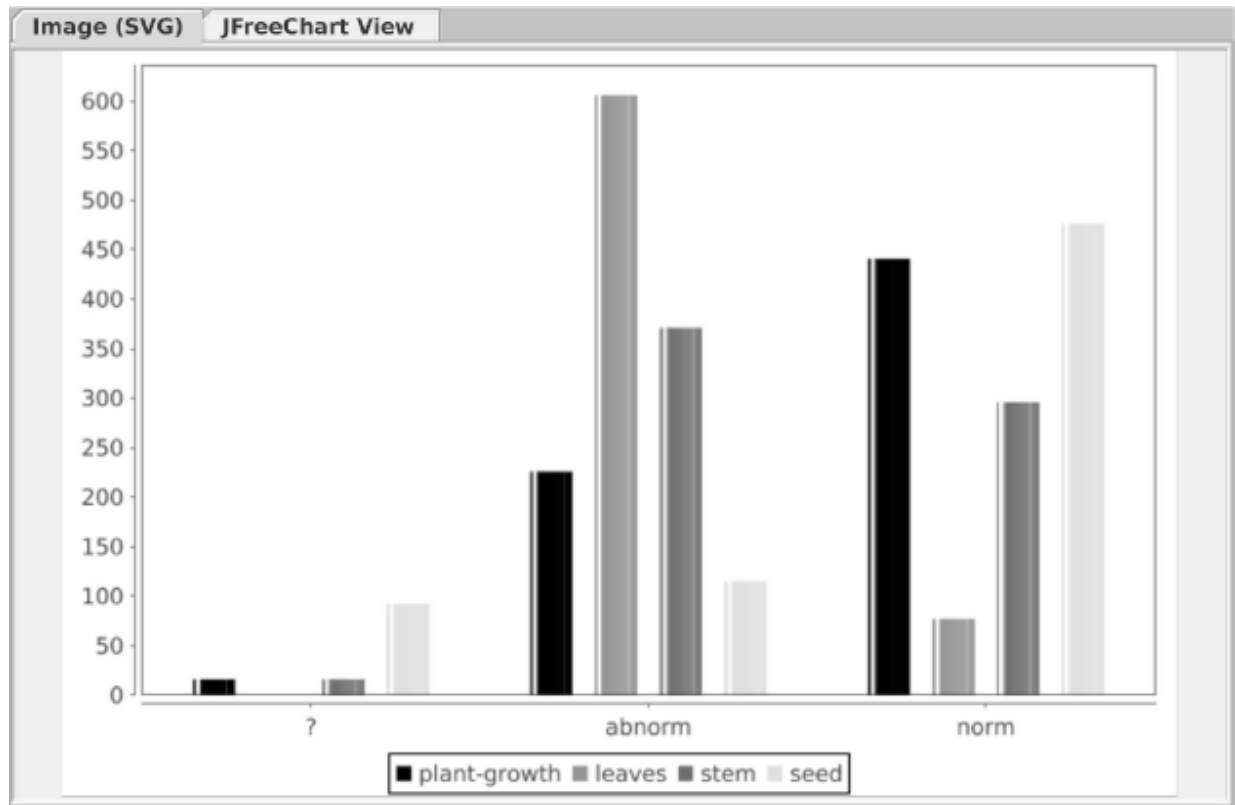


Fig. 9.13 Visualization, bar chart 2

Bar charts are useful not only for nominal and ordinal data, but also for metric data. Bar chart 3 (→ Figure 9.14) shows the Iris data. First, the data is discretized using binning. In this example, ranges of value for the four attributes are divided separately into ten intervals (bins) each. We can see that bar charts reach their limits when displaying multiple attributes, as the bars partially overlap.

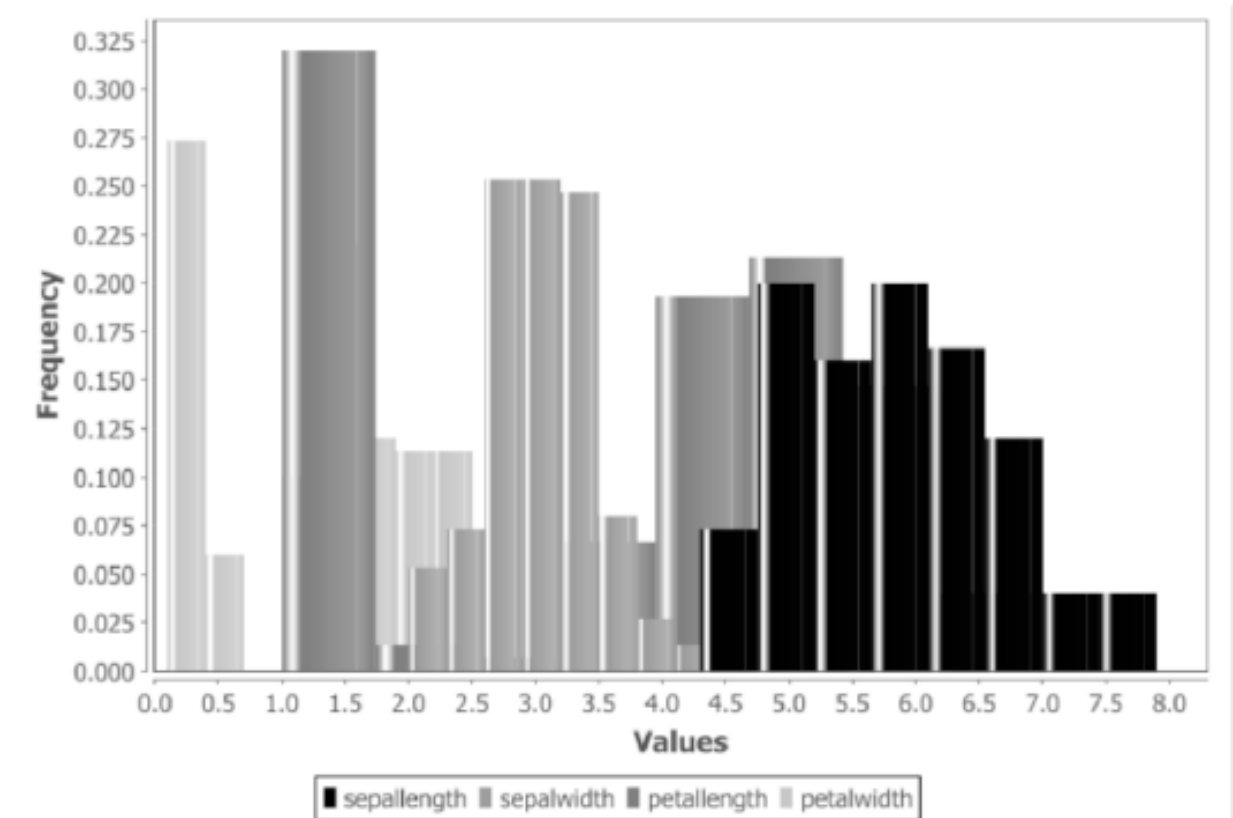


Fig. 9.14 Visualization, bar chart 3

Histograms are a type of bar chart. → Figure 9.15 shows the distribution of classes for *sepal length*. Six intervals were formed for *sepal length*, again using binning. *Iris-setosa* is shown in gray, *Iris-versicolor* in light gray, and *Iris-virginica* in black.

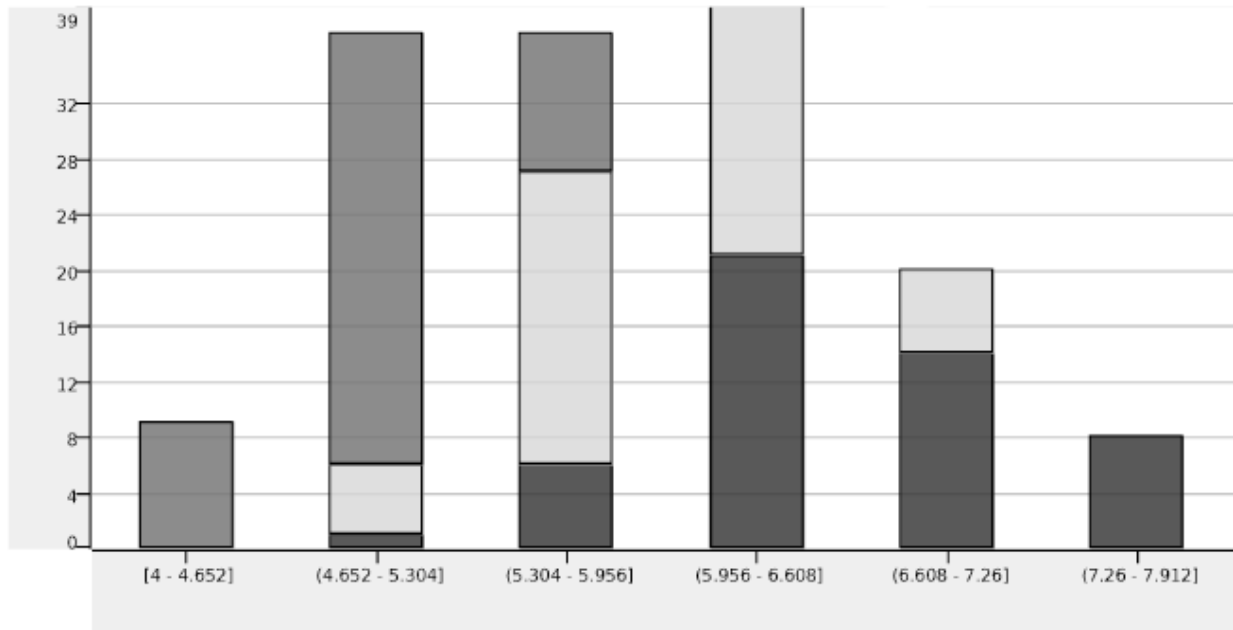


Fig. 9.15 Visualization, histogram

Here, too, a slight correlation between *sepal length* and class assignment can be seen. Smaller values indicate the *Iris-setosa* class, larger values indicate the *Iris-virginica* class.

An alternative to the bar chart is the *pie chart*. → Figure 9.16 shows the distribution of diseases affecting soybeans. The size of the pie segments corresponds to the percentage share of the total amount. Such a pie chart is useful for a presentation, but not for data preparation work.

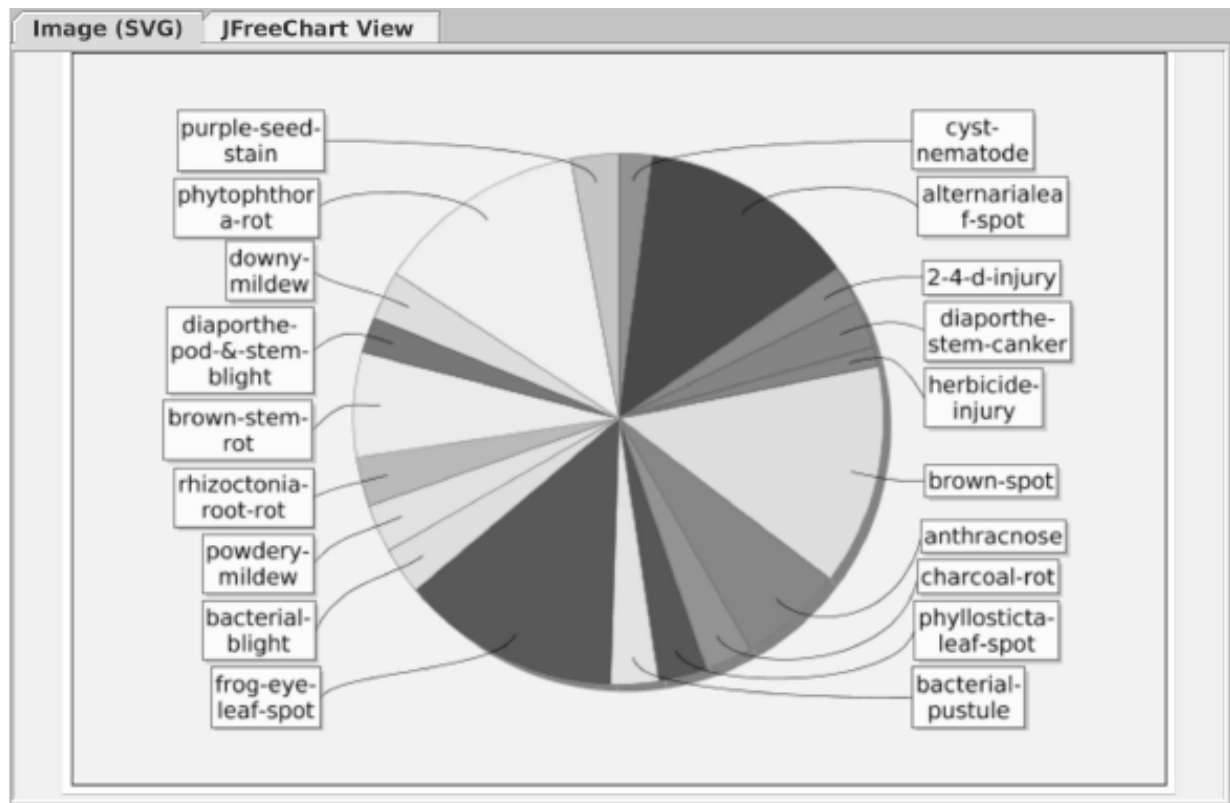


Fig. 9.16 Visualization, pie chart

Box plots have proven useful for visualizing the distribution of values for numerical attributes (→ Figure 9.17). A box plot consists of a rectangle or box, and two lines.

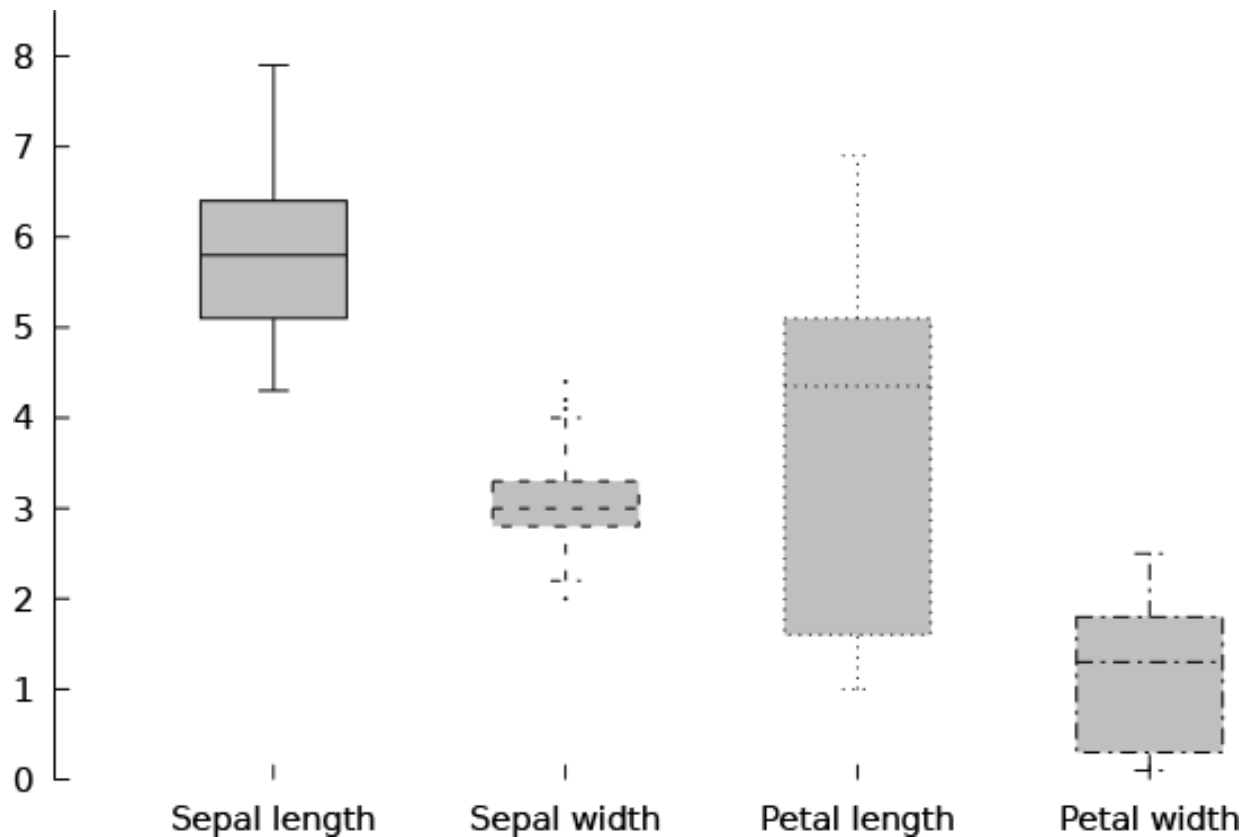


Fig. 9.17 Visualization, box plot

The lines are referred to as “whiskers” and are terminated by a line. The box also contains a line, which is the median of the distribution. The box itself contains the middle 50 % of the data. Below the box are 25 % of the data, and above it are another 25 %.

The chart types discussed so far are suitable for visualizing the given data in the preparatory phases of data analysis. With some limitations, this also applies to pie charts. In particular, the phases of understanding the data and preparing the data (see CRISP-DM model in → Section 1.2) can benefit from good visualizations, for example by helping to formulate initial hypotheses.

Scatter plots are very well suited for visualizing classes and clusters, but they can also be used in data preparation. We have already used a scatter plot in → Figure 9.8. Scatter plots are good for visualizing the distribution of classes, in the case of otherwise numerical attributes. In → Figure 9.18, this has been done for the Iris data set using gnuplot. This two-dimensional representation now shows even four dimensions: *petal length*, *sepal length*, *sepal width*, and the *Iris class*.

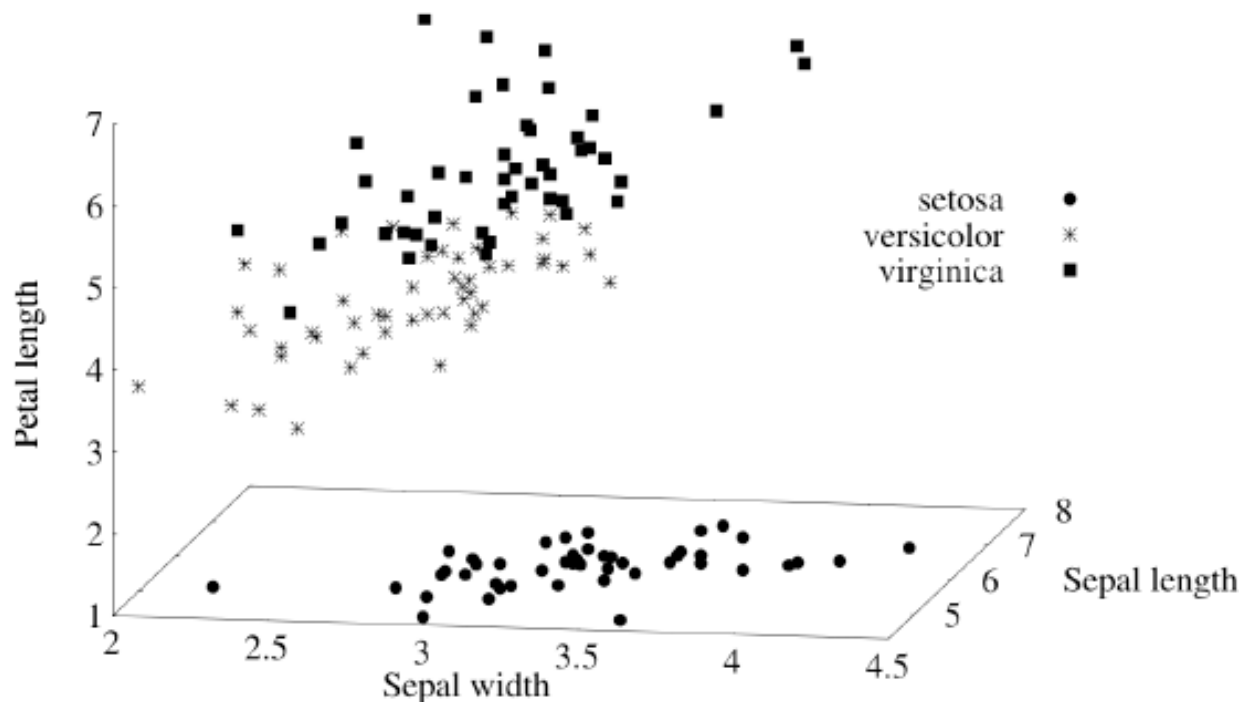


Fig. 9.18 Visualization, scatter plot

Bubble charts are a variation of scatter plots. → Figure 9.19 shows the Iris data set for *petal width* and *petal length*. The classes are indicated by gray values. A fourth dimension can also be visualized in a bubble chart, namely by the size of the bubbles; in this case the values for *sepal length*.

→ Figure 9.19 now shows not only the relationship between *petal width* and *petal length*, but also the relationships between *petal width* and Iris classes, between *petal length* and Iris classes, and between *sepal length* and Iris classes.

Another variant of scatter plots are the *scatter matrix plots* available in KNIME, which combine several scatter plots in one window. → Figure 9.20 shows the Iris data set with regard to *petal length*, *sepal length*, and *sepal width*. The classes are marked in gray.

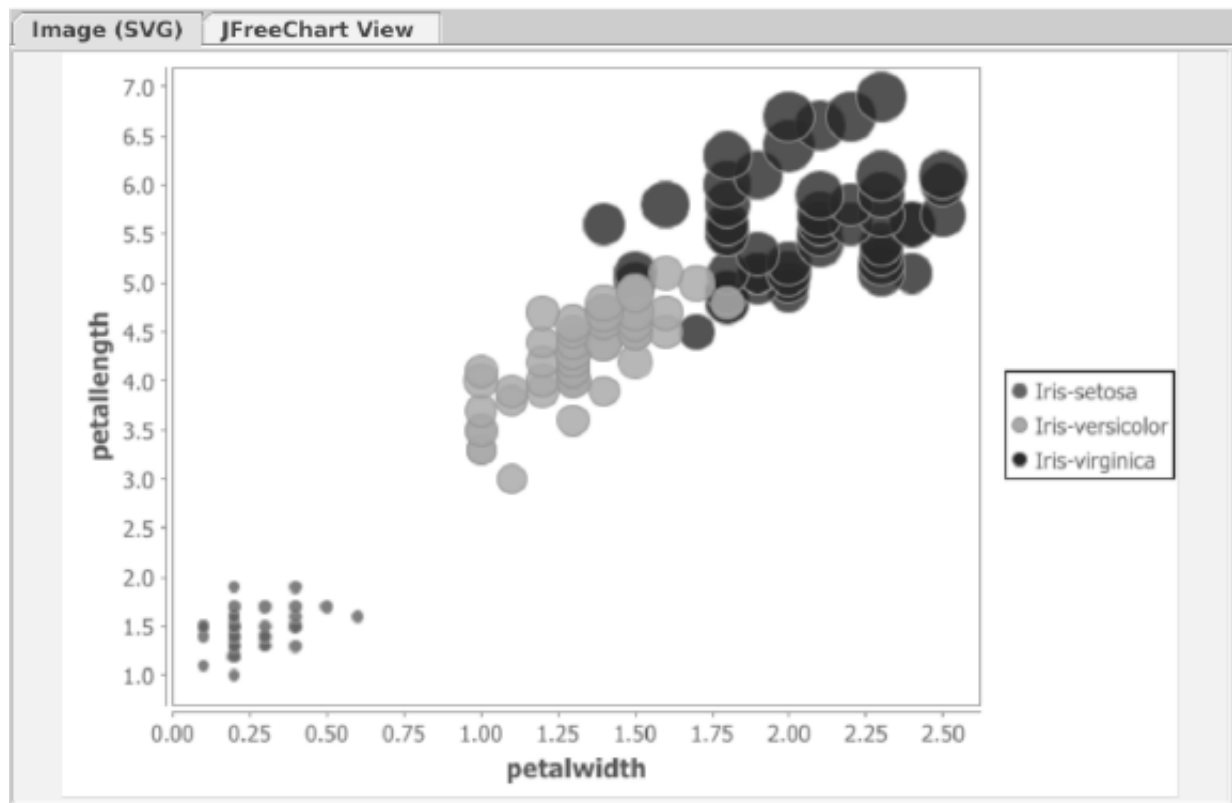


Fig. 9.19 Visualization, bubble chart

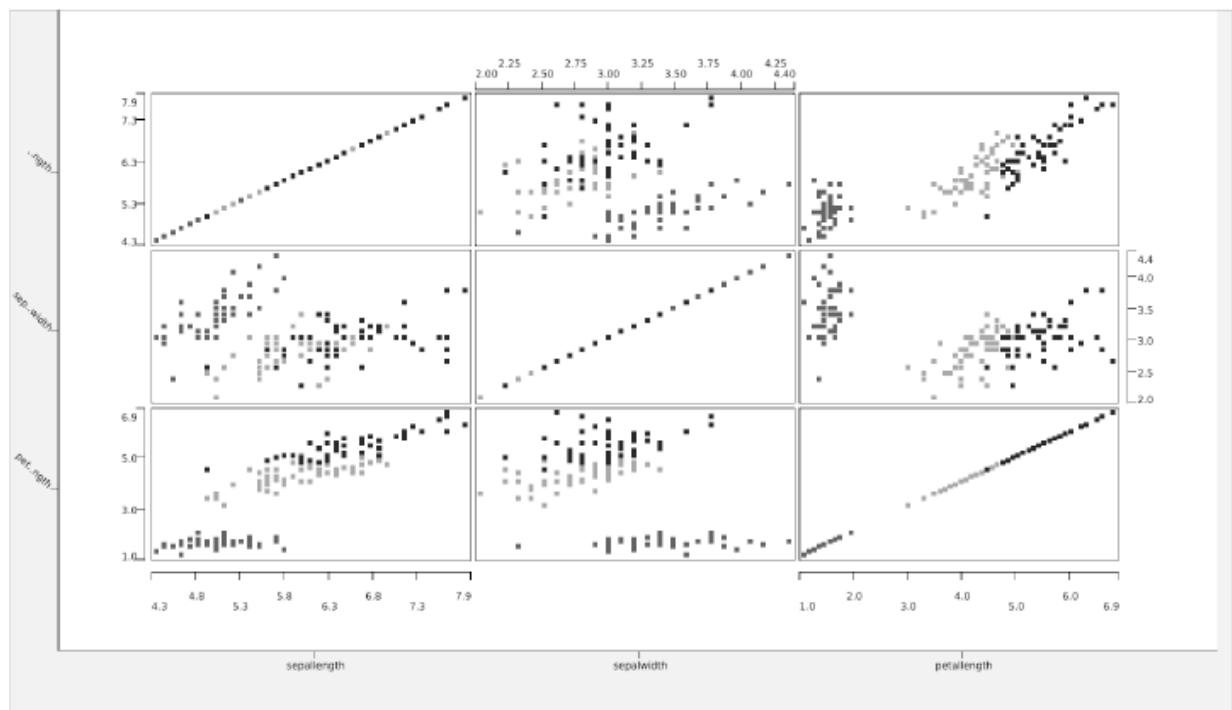


Fig. 9.20 Visualization, scatter matrix

One advantage of scatter plots is that they can display multiple attribute combinations in a single representation.

There are a number of other types of charts. A tree map is used to visualize hierarchical structures. The hierarchy is represented by nested rectangles. The size of the rectangles reflects the proportions. This is used, for example, in hierarchical clustering. There are other variants of bar charts, such as stacked column charts and staircase charts.

A good introduction to the different types of visualization is provided in → [Yau24].

However, every visualization reaches its limits as the dimension of the data increases. Although we have seen that four attributes can be visualized simultaneously, or even five attributes if color and size are used, it is usually difficult to limit the relevant information to four dimensions. In the soybean example (→ Appendix A.2), we have more than 30 attributes, and this is fairly typical. In this respect, the desire to reduce the dimension is justified. The dimension can be reduced in various ways. The classic approach is *principal component analysis* (PCA), which requires numerical attributes. PCA modifies the data in such a way that its structure is reproduced parallel to the axes. We have already discussed this technique in the context of data preprocessing (→ Section 8.2.5, p. 221).

It can also be helpful in visualizing data. If we look at the Iris data, only considering *petal length* and *sepal length* and normalized to the interval [0, 1], as shown in → Figure 9.21 (left), we see that a slight clockwise rotation leads to a better representation of the data with respect to the x-axis. As shown in → Chapter 8, all data records x are transformed using a PCA transformation matrix:

$$x_{new} = \text{PCA} \cdot (x - \bar{x}) \quad 9.13$$

The vector $\bar{x}$ contains the mean values of the x components.

Performing principal component analysis on the Iris data yields the data shown in → Figure 9.21 (right).

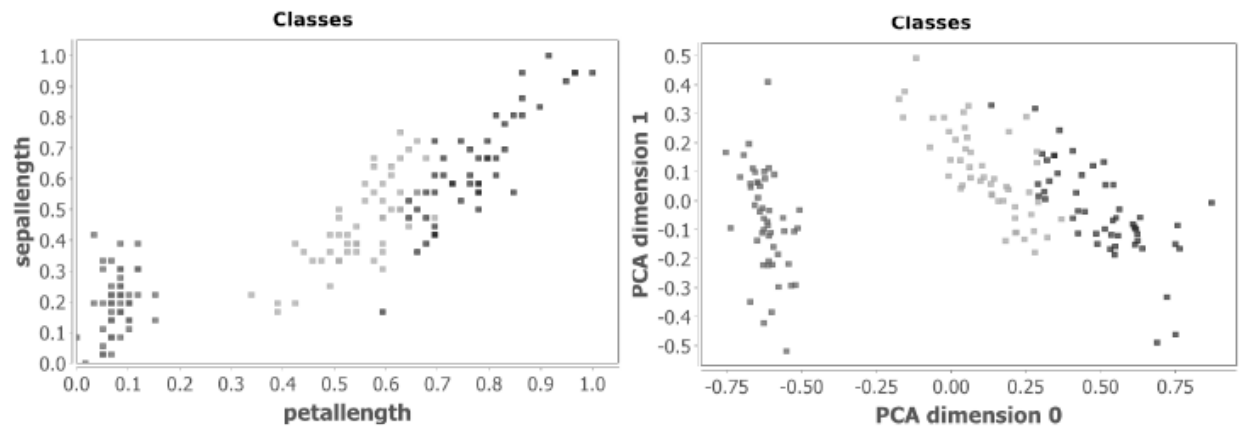


Fig. 9.21 Iris data before and after PCA transformation

10 A data mining task

The simpler it looks, the more problems it hides.

Lee's Law of Electrical Repair

10.1 The task

The examples discussed so far in this book serve primarily to illustrate approaches or methods and their respective mathematical backgrounds. However, data mining is not applied to data sets consisting of only a few dozen data records. Realistic tasks may occasionally be based on a few hundred data records, but typically they involve analyzing thousands or even millions of data records.

In this chapter, we illustrate the data mining approach using a somewhat more comprehensive example. For many years, PRUDSYS AG in Chemnitz, Germany has been organizing the Data Mining Cup, in which a data mining task has to be completed each year. The data for this task is provided by companies, so that the Data Mining Cup can be considered a good test for real-world problems.¹

We refer back to the task from 2002, which involved developing a prediction model for a mailing campaign by an energy supplier. This example may be old, in our fast-paced world, it is probably ancient, but firstly, it is practical for our purposes. Secondly, the methods presented in the book can be easily demonstrated using this example, and thirdly, classification of customers is a typical data mining task.

This chapter is based on the process model for data mining tasks introduced in → Section 1.2. Possible approaches and experiments are discussed, covering the entire process from problem analysis and understanding the data to data preparation, the use of data mining algorithms, and evaluation. This chapter therefore serves to revisit data mining and review the individual steps using a more complex task. The explanations can be used as a guide for working on your own problems.

The Data Mining Cup provides training data that can be used to develop and train a prediction model. The model developed is then applied to a second set of data, the classification data. The resulting predictions are submitted as part of the competition. The real class assignments for the classification data are only published after the competition has ended.

Although we have the results for the classification data for the selected example, we will proceed as far as possible within the scope of the competition and only look at the correct classification results at the end.

- The training data is used both to train and to test the developed prediction model.
- Only then will the classification data be classified using the developed model and this result is compared with the real result data.

This allows the approach and its results to be compared with the top scores from the 2002 competition. Some solutions and the associated data can be found on the book's website, → <https://www.fww.hs-wismar.de/datamining/>.

10.2 The problem

In the open market for energy suppliers, customers frequently switch providers. Based on its experience in previous years, one company in the industry has determined that 10 % of its customers switch providers each year. Can this historical data be used to develop a prediction model that provides information about which customers are likely to cancel their contracts in the current year? Customers who are classified as potential *cancelers* based on the prediction will then receive an offer. With this offer, the company hopes to convince customers not to switch the energy supplier. The following monetary amounts were determined from a business analysis of the problem:

- With a customer who does not cancel, the company earns € 72.00.
- A customer who cancels, does not generate any further revenue for the company.
- If customers who wish to cancel are offered a deal, in some cases it is possible to prevent them from canceling. This generates an average revenue of € 43.80 per customer.
- If a customer who did not intend to cancel receives an offer, they will accept this more favorable offer. In this case, the company will only earn € 66.30, meaning it incurs a small loss.

The evaluation matrix shown in → Table 10.1 summarizes the results.

Tab. 10.1 Customer evaluation matrix

Customer wants to cancel	yes	no
Customer receives offer	€ 43.80	€ 66.30
Customer does not receive an offer	€ 0.00	€ 72

The revenue generated without a discount campaign is

$$f_{base} = € 72.00 \cdot \text{number of non cancelers}$$

The goal is to develop a prediction model that, in preparation for the discount campaign, classifies customers as accurately as possible into two groups: those likely to cancel (*cancelers*) and those likely to remain loyal (*non-cancelers*). A good prediction must achieve a better result than would be possible without the discount campaign. The *objective function* describes the additional profit that can be expected from the discount campaign. The prediction must increase this additional profit as much as possible.

$$\begin{aligned} profit &= € 43.80 \cdot \text{number of identified cancelers} \\ &+ € 72.00 \cdot \text{number of identified non – cancelers} \\ &+ € 66.30 \cdot \text{number of unidentified non – cancelers} \\ &- f_{base} \end{aligned}$$

We will use the following abbreviations for these quantities which refer to the quality metrics in → Section 9.3.2:

- *cc*: Number of correctly identified cancelers (will be contacted), corresponds to TP
- *nc*: Number of unidentified cancelers (will not be contacted), corresponds to FN
- *cn*: Number of correctly identified non-canceling customers TN (will not be contacted)
- *nn*: Number of unidentified non-cancelers FP (will be contacted)

The objective function p , which is to be maximized, can thus be formulated as follows:

$$\begin{aligned}
p &= 43.80 \cdot cc + 66.30 \cdot nn + 72.00 \cdot cn - 72.00 \cdot (cn + nn) \\
&= 43.80 \cdot cc - 5.70 \cdot nn \\
&= 43.80 \cdot TP - 5.70 \cdot FP
\end{aligned}$$

The last term in the first line, $72.00 \cdot (cn + nn)$, corresponds to the base value that the company achieves without a discount campaign. This base value is not added to the additional profit p . The second line shows the compressed form of the objective function.

Given are the training data Tr and a data set K for which classification is to be performed. The sets Tr and K each contain 10,000 data records, 10 % of which belong to the set of potential cancelers. The following estimates can be made for a set of 10,000 customer data records:

1. Revenue generated without a discount campaign:
 $9,000 \cdot 72.00 = \text{€ } 648,000$
2. Revenue generated by a discount campaign in which every customer is contacted:
 $9,000 \cdot 66.30 + 1,000 \cdot 43.80 = 596,700 + 43,800 = \text{€ } 640,500$
3. Revenue that can be expected if classification is 100 % accurate:
 $9,000 \cdot 72.00 + 1,000 \cdot 43.80 = 648,000 + 43,800 = \text{€ } 691,800$

In purely theoretical terms, a prediction model for the discount campaign could therefore generate additional profits of up to € 43,800 per 10,000 customers. In the 2002 competition, the results shown in → Table 10.2 were achieved.

Tab. 10.2 Results in the 2002 Data Mining Cup

Rank	Canceler	Non-canceler	Profit
1.	523	2673	€ 7671.30
2.	477	2415	€ 7127.10
3.	448	2213	€7008.30
4.	440	2156	€6982.80
5.	454	2269	€6951.90
6.	456	2287	€6936.90
7.	379	1731	€6733.50
8.	411	1991	€6653.10
9.	446	2274	€6573.00
10.	418	2059	€6572.10

At first glance, an additional profit of € 7,000 does not seem very high, but this is only the additional revenue per 10,000 customers. Energy suppliers typically have several hundred thousand or even millions of customers, so this represents a significant amount of money that can be generated through data mining.

10.3 The data

The data records of the training and classification sets contain 33 attributes, including the identification number (see → Table 10.3). The training data set also contains the attribute *canceler*, which reflects the experience from the previous year: Who canceled?

The data marked with an asterisk (*) in → Table 10.3 is microgeographic information that was made available for the Data Mining Cup task by an external service provider.

When reviewing the features, the question inevitably arises: What connections could there be between features such as car index, insurance type, or pharmaceutical typology and the classification result, i.e., *canceler* or *non-canceler*? This is a typical question at the beginning of an analysis.

A connection cannot be ruled out from the outset, however unlikely it may seem. Data mining is all about uncovering connections that are not obvious.

In any case, it must be clarified whether data protection regulations allow the available data to be included in the investigation. Only if there are no objections regarding data protection can the data be used for evaluation. For the purposes of this example, we assume that data protection regulations are being complied with. From a content perspective, no features are omitted, but from a technical perspective, it may be the case that some features are not included.

In addition to an initial look at the significance of the features, a statistical analysis of the data is necessary at the outset, as the feature properties influence the type of data preprocessing and the choice of method. The statistics can be determined using a spreadsheet or the statistics node in KNIME.

Tab. 10.3 Features with data type in the energy supplier's data records

Feature	Type	Description
ID	Integer	ID feature, database key
payment_type	Integer	Type of payment: 1 bank transfer, 2 direct debit, 3 other arrangement
power_consumption	Integer	Last annual electricity consumption in kWh
HHH	Integer	Number of households in the building
HGEW	Integer	Number of businesses in the building
MTREG0G	Integer	Regional type*
MTKAU0G	Integer	Purchasing power*
MTSTR0G	Integer	Street type*
MTBEB0G	Integer	Development type*
MTSTA0G	Integer	Status*
MTBON0G	Integer	Credit rating*
MTADE0G	Integer	Proportion foreigners*
MTALT0G	Integer	Age structure*
MTFAM0G	Integer	Marital status*
MTKDI0G	Integer	Car indices: car density*
MTKLE0G	Integer	Car indices: performance index*
MTKKL0G	Integer	Car indices: minibus index*
MTKGB0G	Integer	Car indices: used car index*
MTKGL0G	Integer	Car indices: SUV index*
SCMWGR2	Integer	Insurance type: loyal representative customer*
SCMWGR3	Integer	Insurance type: demanding delegator*
SCMWGR4	Integer	Insurance type: price-oriented rationalist*
SCMWGR5	Integer	Insurance type: overwhelmed support seeker*
SCMWGR6	Integer	Insurance type: skeptical-indifferent*
SCMWGR7	Integer	Insurance type: distinguished-conservative*
SCMWGR21	Integer	Insurance type: affinity for direct marketing*
SCMWGR22	Integer	Insurance type: affinity for direct sales*
PHARM1	Integer	Pharmatypology: healthy muscleman*
PHARM2	Integer	Pharmatypology: uncritical whiner*
PHARM3	Integer	Pharmatypology: Sceptical refuser*
PHARM4	Integer	Pharmatypology: informed body-conscious*
PHARM5	Integer	Pharmatypology: restricted public health insurance patient*
PHARM6	Integer	Pharmatypology: conservative believer in doctors*
canceler	yes/no	Target characteristic: yes = canceler, no = non-canceler

- What data type and value range does a feature have?

- What is the frequency and distribution of the values?
- Are there any data records with missing values?
- Can dependencies between features be identified?

The training data consists of 10,000 data records. The minimum, maximum, mean, and median are determined for the numerical data. In addition, an analysis is performed to determine whether any data is missing and, if so, how much. → Table 10.4 provides information about the available data. The ID column is omitted because all identification numbers are different and therefore cannot contribute to an analysis.

Tab. 10.4 Statistics of the training data

	Min	Max	Average	Median	Missing values
payment_type	1	3	1.737	2	0
power consumption	0	27,500	2,491	2,089	0
HHH	0	82	3.230	2	851
HGEW	0	29	0.290	0	851
MTREG0G	11	16	14.438	15	852
MTKAU0G	-9	-1	-6.367	-7	852
MTSTR0G	1	5	1.832	1	852
MTBEB0G		5	1.598	1	852
MTSTA0G	1	9	4.373	4	852
MTBON0G	1	9	4.010	4	852
MTADE0G	1	9	6.555	7	852
MTALT0G	1	8	4.512	5	852
MTFAM0G	1	9	5.281	6	852
MTKDI0G	1	10	7.168	8	854
MTKLE0G	1	10	2.694	2	854
MTKKL0G		6	1.793	0	2117
MTKGB0G	1	9	4.773	5	854
MTKGL0G		3	0.897	0	854
SCMWGR2	1	5	3.039	3	2296
SCMWGR3	1	5	3.008	3	2188
SCMWGR4	1	5	3.324	4	2296
SCMWGR5	1	5	3.058	3	2286
SCMWGR6	1	5	2.577	2	2188
SCMWGR7	1	5	3.092	3	2285
SCMWGR21	1	5	2.951	3	2296
SCMWGR22	1	5	2.561	2	2295
PHARM1	1	7	4.448	5	2012
PHARM2	1	7	2.240	2	2012
PHARM3	1	7	4.514	5	2012
PHARM4	1	7	3.482	4	2012
PHARM5	1	7	3.571	3	2012
PHARM6	1	7	5.490	6	2012

For the classification feature *canceler*, we determine the number of data records in each class and obtain 9,000 for the class of non-cancelers (no) and

1,000 for the class of cancelers (yes). As expected, the feature *canceler* does not contain any missing values.

If, as in the Data Mining Cup, there is classification data for which no classification result is available, then this data should also be subjected to statistical analysis. A data mining process can only be expected to find a result if this data behaves similarly to the training data. → Table 10.5 provides information and shows that the data is largely within the same value ranges. This is not surprising, since the feature groups MT*, SCHWGR*, and PHARM* in particular are already coded nominal values. Energy consumption is also similar. This means that a fundamental prerequisite for successful data mining is fulfilled.

Tab. 10.5 Statistics of the classification data

	Min	Max	Average	Median	Missing values
payment_type	1	3	1.747	2	0
power_consumption	0	31,062	2,526	2,123.5	0
HHH	0	137	3.206	2	853
HGEW	0	40	0.299	0	853
MTREG0G	11	16	14.434	15	856
MTKAU0G	-9	-1	-6.413	-7	856
MTSTR0G	1	5	1.829	1	856
MTBEB0G		5	1.586	1	856
MTSTA0G	1	9	4.322	4	856
MTBON0G	1	9	3.934	3	856
MTADE0G	1	9	6.560	7	856
MTALT0G	1	8	4.513	4.5	856
MTFAM0G	1	9	5.327	6	856
MTKDI0G	1	10	7.225	8	862
MTKLE0G	1	10	2.701	2	862
MTKKL0G		6	1.749	0	2075
MTKGB0G	1	9	4.849	5	862
MTKGL0G		3	0.891	0	862
SCMWGR2	1	5	3.027	3	2255
SCMWGR3	1	5	2.992	3	2139
SCMWGR4	1	5	3.301	3	2255
SCMWGR5	1	5	3.098	3	2242
SCMWGR6	1	5	2.599	2	2139
SCMWGR7	1	5	3.086	3	2243
SCMWGR21	1	5	2.966	3	2255
SCMWGR22	1	5	2.535	2	2254
PHARM1	1	7	4.454	5	1969
PHARM2	1	7	2.275	2	1969
PHARM3	1	7	4.490	5	1969
PHARM4	1	7	3.489	4	1969
PHARM5	1	7	3.546	3	1969
PHARM6	1	7	5.484	6	1969

Before we proceed with further steps in the data mining process, we will conduct a quick trial and develop an initial classification model without data preparation, if possible. The aim is to gain an initial impression of a possible

result. In addition, initial concrete experiments always lead to a better understanding of the data and also provide inspiration for further action.

A decision tree learning method can be used largely without preprocessing if it can handle numerical values. With KNIME, an initial data analysis process can be developed quite quickly. The necessary steps in the process are:

1. Import the training data
2. Handling missing values
3. Splitting the training data into a file for training and a file for testing
4. Using the node for decision tree learning
5. Calculating the result according to the objective function

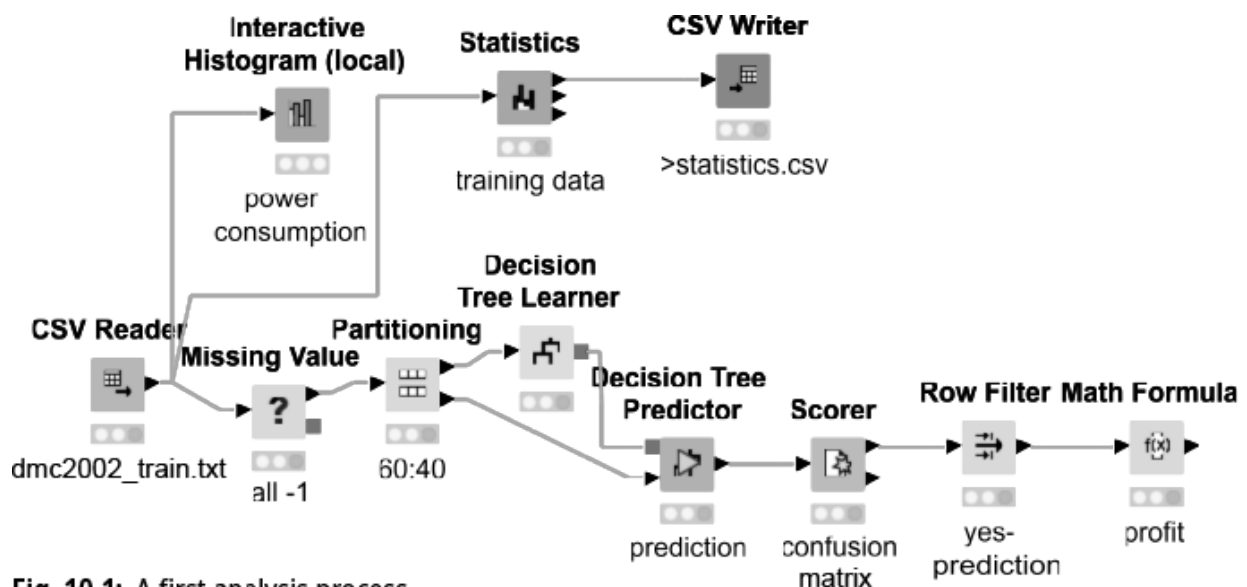


Fig. 10.1: A first analysis process

Fig. 10.1 A first analysis process

→ Figure 10.1 shows the sequence of this first experiment. Two nodes were included here that provide information about the data. One node displays statistical parameters of the features, while the other node provides information about the frequency distribution of the values of the feature energy consumption. Both nodes are not absolutely necessary for the first experiment.

The classification of the test data can be found in the *Prediction (canceler)* node in the two decisive columns: the expected output in the *canceler* column and the classification calculated by the model in the *Prediction (canceler)* column.

The profit achieved is calculated using the *scorer*, which is configured so that the number of predictions for each class can be read from the rows.

The scorer generates what is known as a *confusion matrix*. The first row contains the customers classified as cancelers (TP, FP), while the second row contains the customers classified as non-cancelers (FN, TN).

Prediction \ Canceler?	yes	no
Prediction yes	TP	FP
Prediction no	FN	TN

Prediction \ Canceler?	yes	no
Prediction yes	69	365
Prediction no	331	3235

Dialog - 0:74 - Scorer (Confusion) — □

File

Scorer

Flow Variables

Memory Policy

First Column

\$ Prediction (canceler) ▼

Second Column

\$ canceler ▼

OK

Apply

Cancel

In our experiment, only 69 cancelers were correctly identified, while 365 non-cancelers were incorrectly classified as cancelers, and 331 cancelers were unfortunately not identified. We are interested in the *yes* row here. This is selected in the *Row Filter*. In the *Math Formula* node, we now apply the (reduced) profit function already shown in Equation 10.1, i.e. ($\$yes \cdot (43.80) + \$no \cdot (-5.70)$). The result is not very high, namely € 941.7. We must take into account here that this amount was achieved on only 40 % of the data. For a comparable estimate for 10,000 data records, the result must be extrapolated accordingly and then amounts to € 2,354.25. Compared to the results in → Table 10.2, we perform only moderately well.

You will certainly obtain different results when you carry out these experiments. The 60:40 split of the data is always random and therefore different each time. However, you can also set it up so that the same split is always used. This makes it easier to compare the results of the different experiments. However, the split also has a significant impact on the result, so each new split offers the chance to obtain a better prediction model. In our experiments, we mostly use this constantly changing, random distribution. The conclusion is: never conduct just one experiment, but always several.

Since the data only contains 10 % of cancelers, both the training set (60 %) and the test set (40 %) should have the same ratio. In KNIME, this is achieved through *stratified sampling*.

In our experiment, normalizing the data by decimal scaling also does not change the result. Quick success is not to be expected. We need to take a closer look at the data and use data preparation options.

10.4 Data preparation

Data provided as part of the Data Mining Cup has always undergone several stages of data preparation. This also applies to the task of an energy supplier discussed here. Microgeographic data was added to the energy supplier's data (*ID*, *payment_type*, *power_consumption*). In addition, a selection was made: the two available sets (the training set and the classification set) each constitute a representative sample of all customer data, reflecting the ratio of customers to cancelers.

Since the first decision tree did not produce the desired result, we will now carry out appropriate experiments. Let us examine the statistical values of the training set again (see → Table 10.4) and compile some notable features:

1. There are many incomplete data records. If all data records containing a missing value are removed, only 7,601 data records remain. This is sufficient for training, but how should the incomplete data records be handled in the classification set?
2. Except for the classification feature, all other features are integers. Most attributes have only a small number of different values, the majority of which are single digits:
(0..5), (1..5), (1..7), (1..10).
3. There is a simple classification into two classes: cancelers and non-cancelers.
4. Although → Table 10.4 does not include the standard deviation, a comparison of the mean values or median values with the respective minimum and maximum values shows that the features *power_consumption* and *HHH* are highly dispersed and that there are probably some outliers here.
5. The feature *MTKAU0G* is the only one that contains negative values.
6. *Power_consumption* and *HHH* are metric, while all others are ordinal. The latter cannot be derived from the value ranges, but from the description of the features.

What approach can be drawn from these findings?

1. The two features *power_consumption* and *HHH* must be examined in terms of their distribution, and the values must be grouped. The aim is to achieve a value range similar to that of the other features, for example 0..10.
2. There are several ways to handle missing values. We will perform tests with a global value (−1). Experience from other projects suggests that good results can be expected if a value is individually determined for each feature. To this end, the missing values are replaced by the median or mean value of the respective feature.
3. Depending on the method, the values can or must be normalized.
4. Since the influence of microgeographic data can be questioned in terms of content, experiments with reduced data records are also interesting.

We will conduct experiments with grouped data for the features *power_consumption* and *HHH*. The starting point is the consideration that the median and mean do not quite “fit” the minimum and maximum. Using a spreadsheet or the corresponding node in KNIME, a histogram is generated that shows the distribution of the data, see → Figure 10.2.

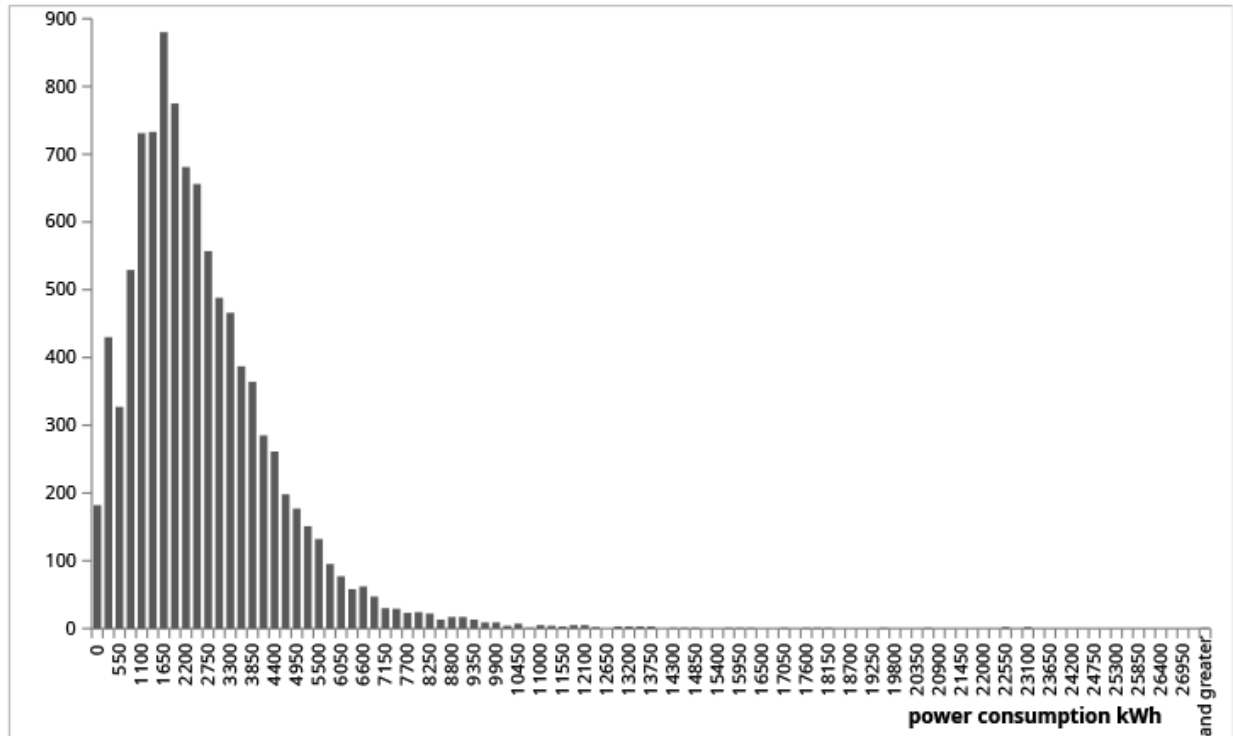


Fig. 10.2 Distribution of power consumption values

While there is a concentration in the range up to around 6,000 kWh, higher values are only found in rare cases. Moreover, it must be anticipated that even some higher values may be found in the classification data. The value range is divided into sub-ranges (*binning*). → Figure 10.3 shows one a possible division. Alternatively, in our experiments, the following mapping to the range 0..5 is used:

```
if($power_consumption$ >5000) return 5;
    else return $power_consumption$/1000;
```

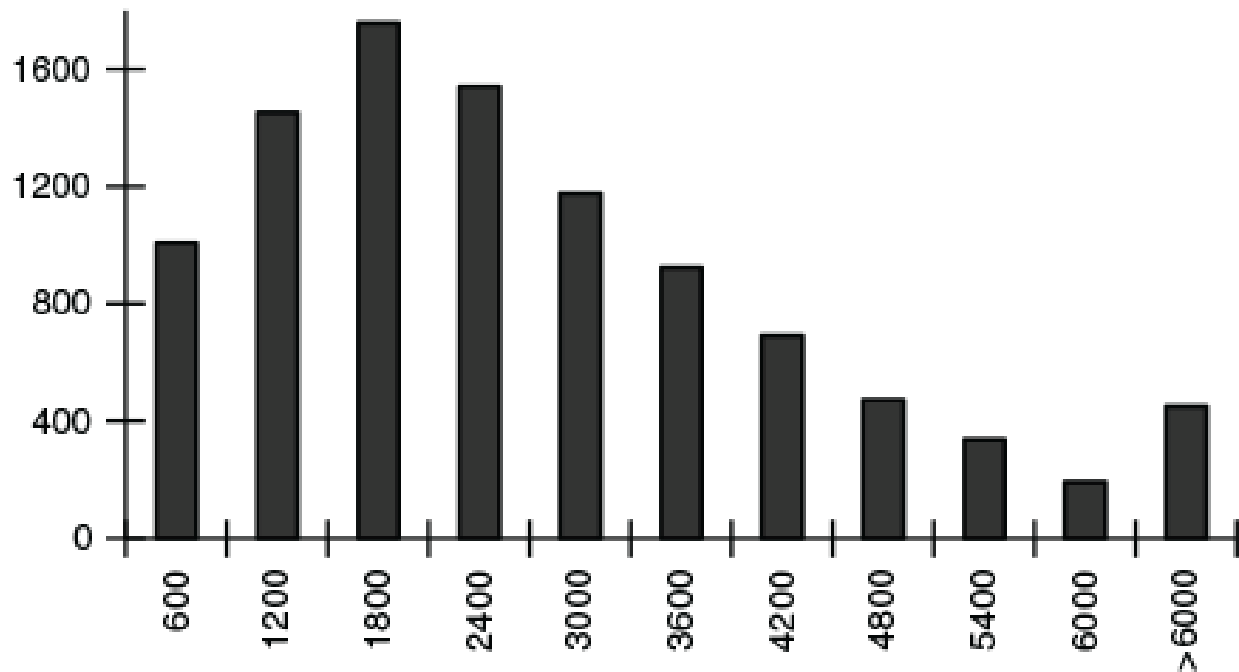


Fig. 10.3 Distribution of power consumption values after binning

The same procedure is used for the characteristic *HHH*, number of households in the building. The histogram (→ Figure 10.4) shows a decreasing accumulation, with values >10 no longer occurring completely and only in very small numbers.

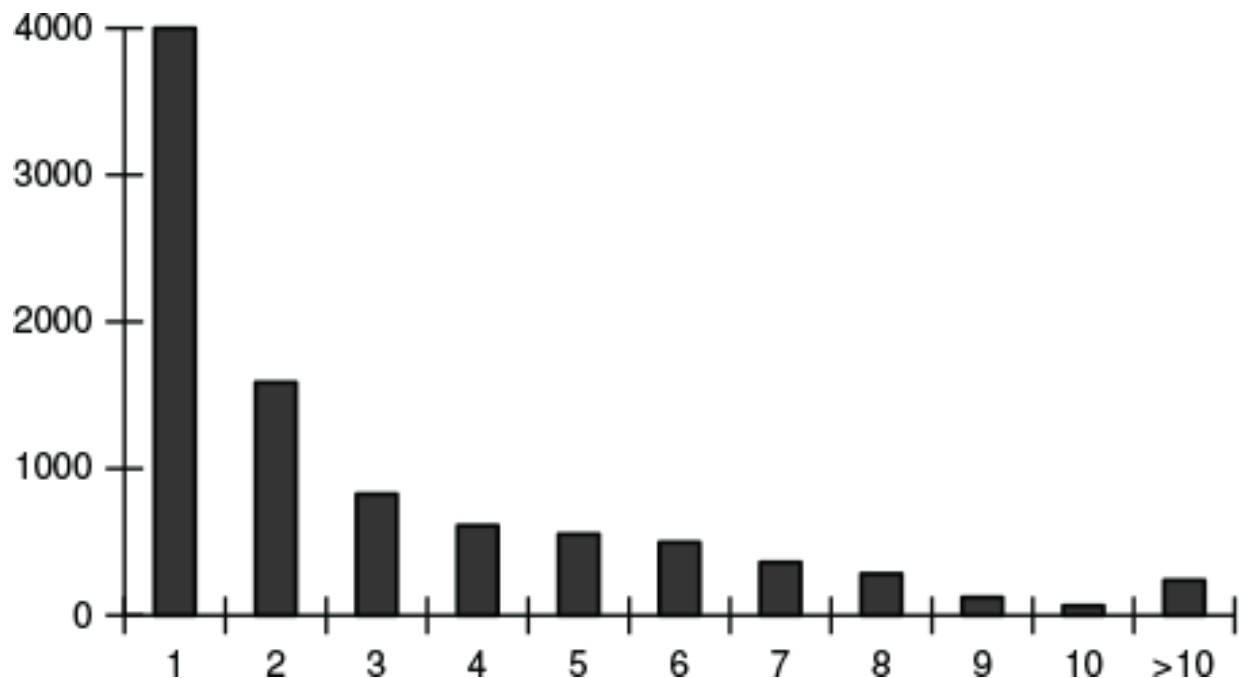


Fig. 10.4 Distribution of HHH values after binning

The maximum HHH value of 82 can certainly be considered an outlier. The value range is divided into the bins 0 to 10 and >10. The latter is coded as 11, so that the range that needs to be processed now only consists of the values 0 to 11.

10.5 Experiments

From the multitude of possible experiments, we are pursuing the approaches listed below as examples. These differ in terms of the data set used, the coding, the handling of missing values, the standardization, and, of course, the method used:

- Features used:
all or only the first three, i.e., *payment_type*, *power_consumption*, and *HHH* (number of households in the building);
- Type of coding for *power_consumption* and *HHH*:
no further coding or binning;
- Type of normalization:
none, min-max, Z-normalization, or decimal scaling;
- Method used:
k-nearest neighbor, naive Bayes, decision tree method, neural networks;

- Software used for neural networks: KNIME or JAVANNNS.
All other experiments are performed in KNIME.
- In addition to the experiments carried out in this section, we present Python-based solutions in → Section 10.6.

Having gained an initial overview and carried out a first experiment, we now need to proceed systematically. It is also essential to keep detailed records of every experiment, no matter how small: the methods used and their parameters, the data and their coding, and, of course, the results obtained. The temptation to try things out is quite high, and one always believes that the next small change will bring about the decisive breakthrough. Even quite successful experiments are quickly forgotten and can then only be repeated with great difficulty and effort.

Let's now compile a list of the experiments to be carried out. This list includes the method, the amount of data, the coding of the features *power_consumption* (*pc*) and *HHH*, the treatment of missing values, the normalization, and the software used. In addition, the preprocessing of the data according to the specific procedure is given. We always start with the configuration listed and modify it depending on the results obtained.

1. K-nearest neighbor; all features; no coding of *pc* and *HHH*, median of the column for missing values; decimal scaling; KNIME.
2. Naive Bayes; all features; *pc* and *HHH* binned; median of the column for missing values; no normalization; KNIME.
All values are converted to nominal values (character strings).
3. Decision tree; all features; no coding of *pc* and *HHH*; the value -1 is used globally for missing values; no normalization; KNIME.
4. Decision tree; all features; no coding of *pc* and *HHH*; median of the column for missing values; normalization (all three normalization methods are tried); KNIME.
5. Decision tree, all features; *pc* and *HHH* are grouped; median of the column for missing values; normalization (all three normalization methods are tried); KNIME.
6. Decision tree; only the three features *payment_type*, *power_consumption*, and *HHH*; *pc* and *HHH* are binned; median of the column for missing values; normalization (all three normalizations are tried); KNIME.
7. Neural network; all features; with and without coding of *pc* and *HHH*; missing values are set to -1; decimal scaling; KNIME.
Two different hidden layers are tried.

8. Neural network; all features; with and without coding of *pc* and *HHH*; missing values are set to -1; decimal scaling; JAVANNNS. Two different hidden layers are tried. The influence of the learning method and the learning factor are investigated.
9. Neural network; only the three features *payment_type*, *power_consumption*, and *HHH*; *pc* and *HHH* are binned; median of the column for missing values; decimal scaling; KNIME.
10. Neural network; only the three features *payment_type*, *power_consumption*, and *HHH*; *pc* and *HHH* are binned; median of the column for missing values; decimal scaling; JAVANNNS.

In the experiments, only the given training set is used to develop a prediction model, i.e., a classifier. In order to compare the models with the results of the Data Mining Cup 2002, the classification data is analyzed and the classification results are compared with the actual, real classes.

10.5.1 K-nearest neighbor

Experiment 1 uses the k-nearest neighbor algorithm (see → Section 5.1). The process is shown in → Figure 10.5.

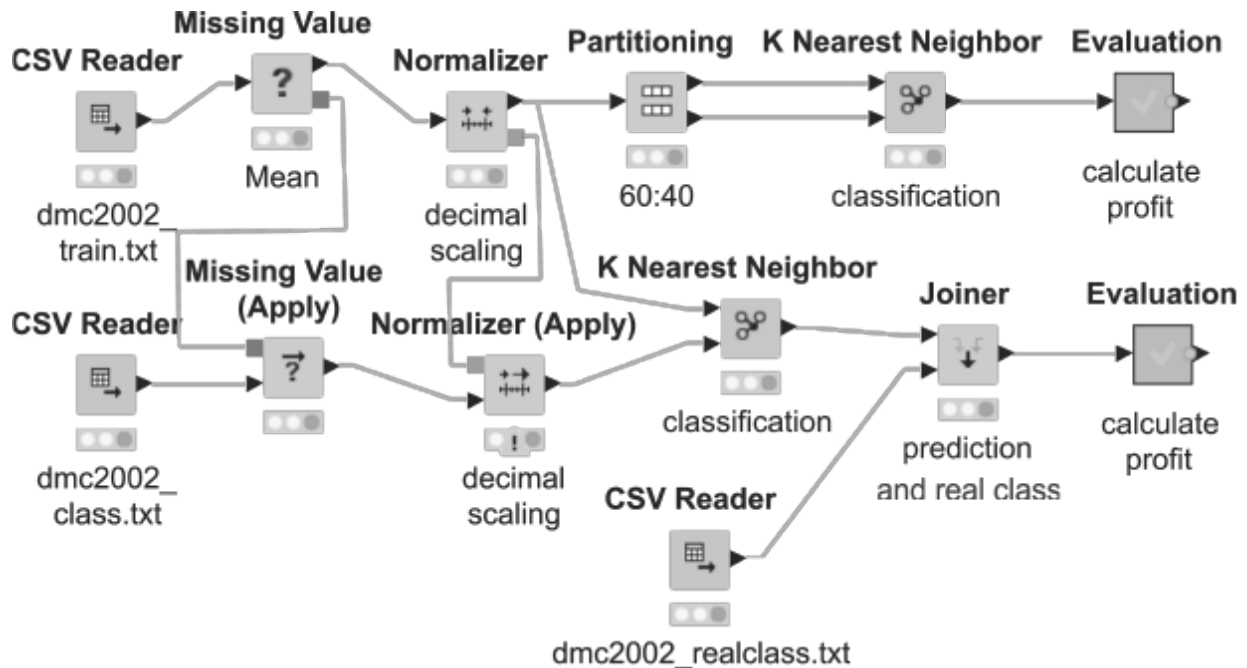


Fig. 10.5 The k-nearest neighbor experiment

The missing values are replaced by the median of the column, and then decimal scaling is performed (normalization to the range 0..1). The provided *data_dmc2002_train* is then divided into a training set (60 % of the data records) and a test set (40 %). The node is configured so that both sets contain the same proportion of data records from a class. In the example, this is 10 % cancelers and 90 % non-cancelers.

After classification, the result is determined. As already presented and used in → Figure 10.1, this is done using the scorer. These nodes are combined here and in all further KNIME experiments into a *meta node (evaluation)*.

For $k = 3$ without weighting the distance for the training set, a value of € -25.20 is obtained.

For $k = 3$, but with weighted distance calculation, € 906.00 is achieved.

For $k = 5$ and with weighted distance calculation, the result is € 111.30 for the test set and € 986.40 for the entire data set to be classified.

These results all refer to the test set, which contains only 40 % of the data records in the provided training set.

Why does the k-nearest neighbor method deliver such poor results? On the one hand, although we normalize the attribute *power_consumption*, the few high values, which then become values equal to or close to 1, shift the apparently important, smaller power consumptions closer to 0, so that the distance between them becomes very small. These are then obviously included in the distance to a much too small extent. On the other hand, in this experiment, using the median as the value for the missing values is probably not a good idea after all. We encode the missing values as -1. So let's modify our experiment:

- K-nearest neighbor; all features; coding of *pc* and *HHH*; -1 for missing values; decimal scaling; weighted distance; KNIME.

The encoding of *pc* and *HHH* is done in such a way that *pc* is capped at 5,000 and *HHH* at 11.

```
if($power_consumption$>5000) return 5;  
    else return $power_consumption$/1000;  
if ($HHH$>10) return 11; else return $HHH$;
```

However, with this configuration, we still only achieve an average of € 700.

Let's take another look at the data. Only 10 % of the data are *cancelers*. This could be the reason why too many predictions favor *non-cancelers*. Let's modify our experiment once again and reduce the number of data records

with *non-cancelers* to 1,000. This can be achieved in KNIME using the *Equal Size Sampling* node. This experiment also yields rather disappointing results. Obviously, not all attributes are equally important for the prediction. However, scaling causes all attributes to be weighted equally in the prediction. We should therefore try again with a reduced set of attributes. To do this, a *column filter* node is inserted into the data preprocessing, and only the columns of the three characteristics and the classification feature *canceler* are included. Based on the sequence shown in → Figure 10.5, the node can be inserted before the *Missing Value* node, for example.

- K-nearest neighbor; only the three features *payment_type*, *power_consumption*, and *HHH*; encoding of *pc* and *HHH*; -1 for missing values; equal size sampling; decimal scaling; weighted distance; KNIME.

With *k* values from 3 to 10, we now achieve values between € 2,000 and € 3,000. You can even achieve over € 5,000 if you discretize the attributes *power_consumption* (0..3) and *HHH* (0..6) as integers.

```
if($power_consumption$ >5000) return 3;
    else return $power_consumption$/2000;
if ($HHH$ > 10) return 6;
    else return $HHH$/2;
```

However, the tests show that the results have a large variance. In particular, the model that works well on the training data is not always the best on the prediction set. An obvious assumption is that the training set is too small and, as a result, unstable solutions are generated by the random distribution. In fact, *equal size sampling* results in a subset that contains only 2,000 data records, since there are only 1,000 cancelers and only 1,000 of the 9,000 non-cancelers are selected to balance the ratio between cancelers and noncancelers. For training, we only use 60 %, i.e., only 1,200 data records.

It is therefore worth considering performing tests with a separate strategy for forming the training set. To do this, the given 10,000 data records can first be divided into *cancelers* and *non-cancelers*. A certain percentage is taken from the cancelers and another percentage from the non-cancelers, so that although an equal ratio is not necessarily achieved, the clear predominance of non-cancelers is reduced. In KNIME, this can be easily implemented with the *Row Splitter*, *Partitioning*, and *Concatenate* nodes, as shown in → Figure 8.5 (p. 212).

If further tests are carried out using this approach, the variance is drastically reduced and stable results above € 6,000 are obtained for the classification set. The best model on the training data is now also the best on the prediction data. The following table shows the best result achieved with k-nearest neighbor.

Result	Characterization of the experiment
6,533.10	k-nearest neighbor, k=7, only 3 attributes
	decimal normalized, no weighted distance, missing values: -1
	coding of <i>power_consumption</i> (discrete): 0..3
	coding of <i>HHH</i> (discrete): 0..6
	separate training set: 800 cancelers, 1000 non-cancelers

10.5.2 Naive Bayes

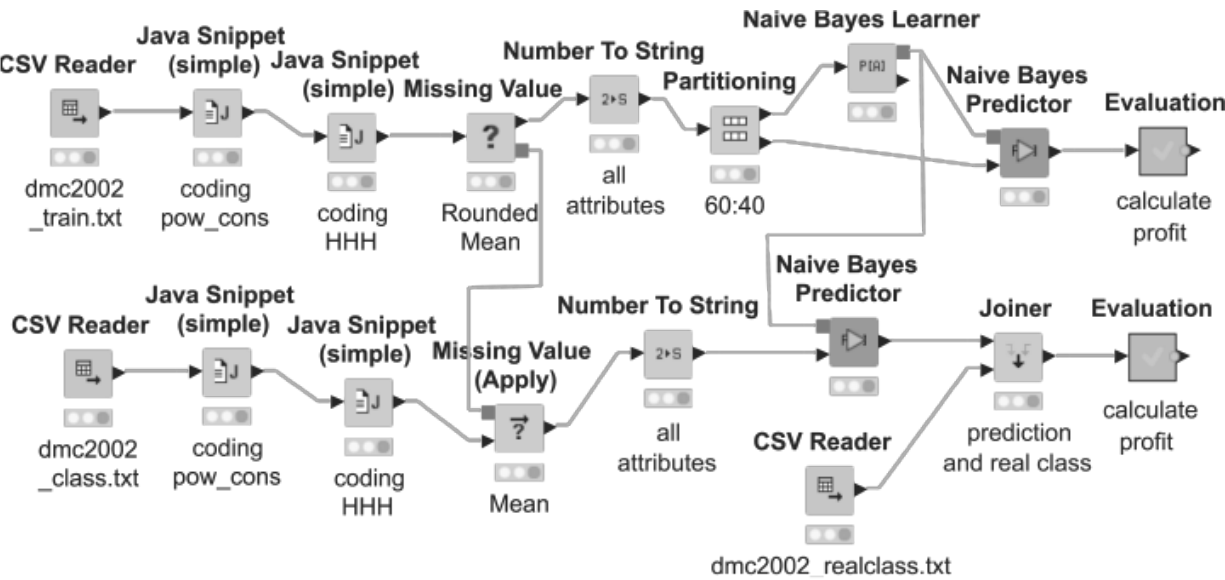


Fig. 10.6 The naive Bayes experiment

Experiment 2 uses the naive Bayes algorithm, see → Section 5.3. The structure is similar to the previous experiment, but more preparatory steps are required here. The values of the features *power_consumption* and *HHH* are grouped as described above. For this purpose, two Java code nodes are inserted, which perform the encoding of the features *power_consumption* and *HHH* already used in k-nearest neighbor:

```

if($power_consumption$>5000) return 5;
    else return $power_consumption$/1000;
if ($HHH$>10) return 11; else return $HHH$;

```

The missing values are first replaced by the (rounded) mean value of the column. Then all values are converted into character strings. Again, 60 % of the training data is used for the learning process and the model is tested using the other 40 % of the data. The learned model is then also used for the classification data of the competition. The results are calculated again as described above.

We obtain a value of € 489.60 for the 40 % test set. For the entire classification set, the value rises to over one thousand, namely to € 1213.80.

Following the considerations in Experiment 1 using k-nearest neighbor, we should carry out further experiments here as well. With naive Bayes, it also seems appropriate to create an equal distribution of the data records with regard to the class attribute (*equal size sampling*). Likewise we should replace the missing values with the value –1. The binning is done as described above (*power_consumption* 0..5, *HHH* 0..11).

With all attributes, we achieve values between € 2,000 and € 5,000 in this experiment. If we only take the three features *payment_type*, *power_consumption*, and *HHH*, we even achieve values above € 6,000. Values between € 4,000 and € 6,500 are achieved with the compact discretization *power_consumption* (0..3) and *HHH* (0..6).

Result	Characterization of the experiment
6,355.50	naive Bayes, only 3 attributes
	missing values: –1
	coding of <i>power_consumption</i> (discrete): 0..5
	coding of <i>HHH</i> (discrete): 0..11
	equal size sampling
6,297.90	as before
	coding of <i>power_consumption</i> (discrete): 0..3
	coding of <i>HHH</i> (discrete): 0..6

However, these results must again be viewed with caution, as the results show high variance and the correlation between the quality of the test data and the quality of the prediction data is low.

Finally, we also carry out experiments with naive Bayes based on the training set we generated ourselves (compare the experiments with k-nearest neighbor). The variance of the results is drastically reduced. We obtain stable results above € 6,000. The best prediction on the test data also achieves the best result on the prediction data. With this approach, we achieve the following results:

Result	Characterization of the experiment
6,165	naive Bayes, only 3 attributes
	missing values: -1
	coding of <i>power_consumption</i> (discrete): 0..3
	coding of <i>HHH</i> (discrete): 0..6
	separate training set: 800 cancelers, 1000 non-cancelers

10.5.3 Decision tree method

Using the decision tree learning method, we will conduct several experiments (Experiments 3 to 6). In doing so, we will again use a data mining process similar to that described in the previous experiments. We start with the process shown in → Figure 10.1 and add, in particular, the classification of the data from *data_dmc2002_class*, see → Figure 10.7.

In the first experiment (Experiment 3), all missing values are replaced by -1 and no normalization is performed. A value of € 407.70 is achieved for the test set, and the result for the entire classification set is € 562.50. In the second attempt, all features are normalized, and we obtain the values € 1063.20 and € 697.80, respectively. This suggests that the decision tree is too tightly fitted to the training set and therefore cannot classify the data in the classification set well.

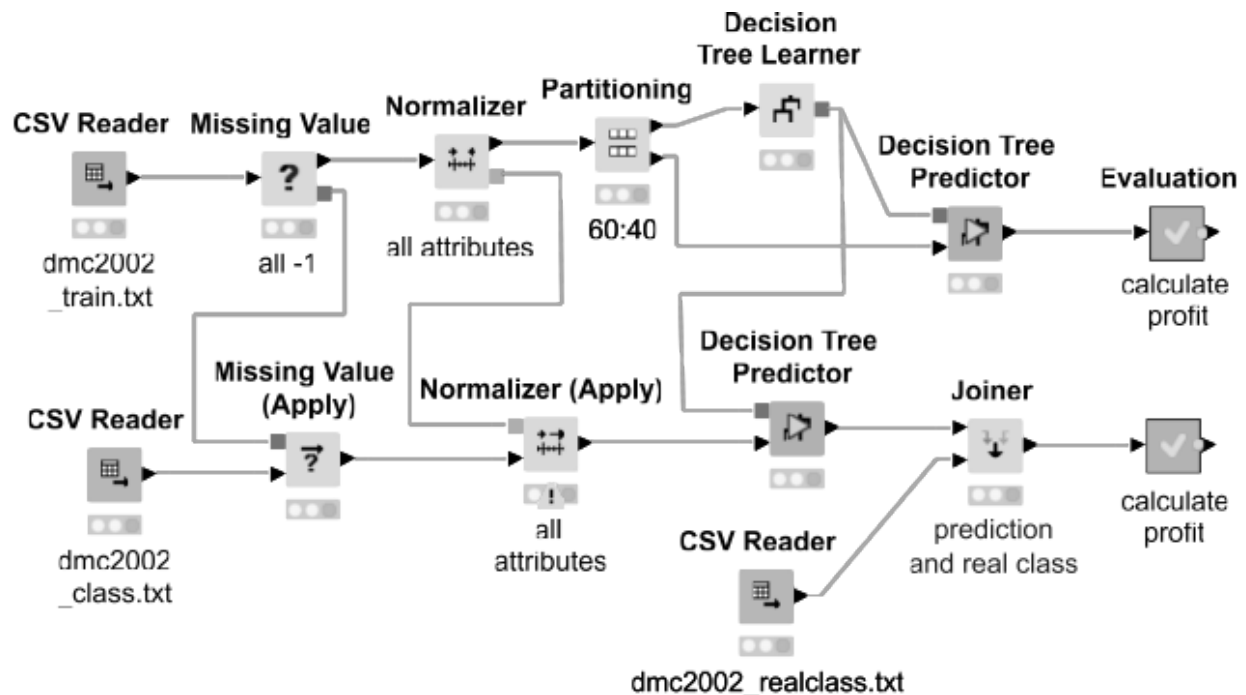


Fig. 10.7 The decision tree learning experiment

In the next experiment, Experiment 4, all missing values are replaced by the median of the respective column. The data is then used first without and then with normalization. → Table 10.6 shows the results.

Tab. 10.6 Results of decision tree learning

Normalization	Test data (4,000)	Classification data (10,000)
no normalization	493.8	409.5
min-max normalization	446.7	479.7
Z normalization	246.9	1290.6
decimal scaling	259.5	1861.5

In Experiment 5, we will now group the features *power_consumption* and *HHH* as described above and again use KNIME's decision tree learning method, applying the various normalizations once more.

The modified data preprocessing procedure corresponds to the procedure shown in → Figure 10.8. However, we initially use all attributes. All further steps correspond to the previous experiments. The results can be found in → Table 10.7.

Tab. 10.7 Results of decision tree learning with encoded power consumption

Normalization	Test data (4,000)	Classification data (10,000)
no normalization	366.6	75.0
min-max normalization	318.3	1558.5
Z normalization	796.5	888.6
decimal scaling	376.8	1377.9

We have not set a reproducible distribution in the partitioning node. This means that with each reset, a random new split is made between the training set (60 %) and the test set (40 %). This can be exploited in the experiments by repeatedly resetting the *partitioning* node to provoke new distributions and thus new results. The data in these tables are therefore only approximate value, which may not necessarily be achieved exactly in your own experiments.

In the next experiment with decision tree learning, the data set is again reduced to the three features *payment_type*, *power_consumption*, and *HHH*, see → Figure 10.8.

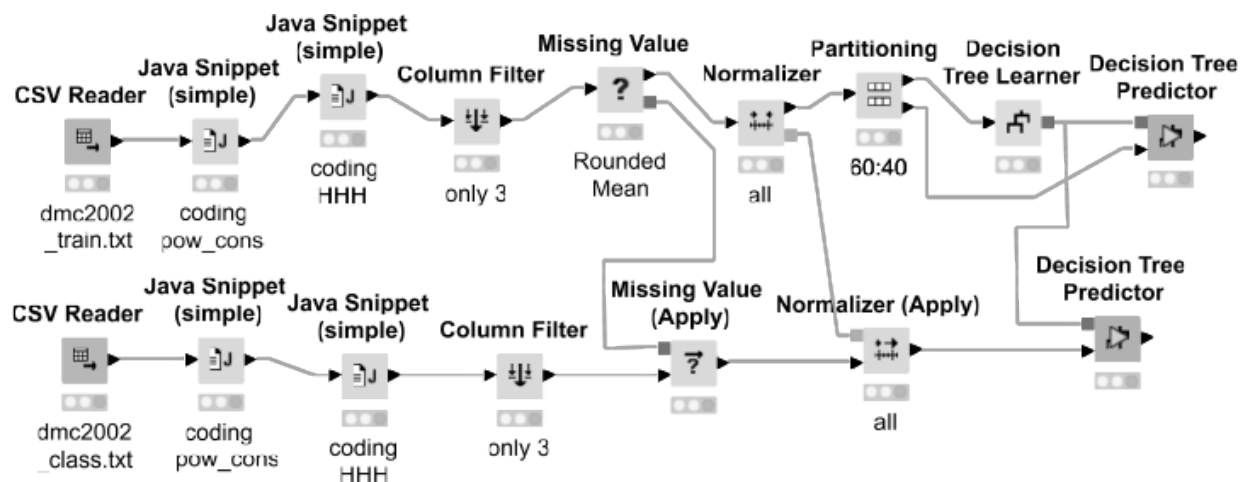


Fig. 10.8 Binning of power_consumption and HHH and reduction to three attributes

The reduction means that the steps leading to the result are processed noticeably faster. However, the results in → Table 10.8 show that this data reduction does not bring the hoped-for success, at least with the chosen method.

Tab. 10.8 Results of decision tree learning with only three features

Normalization	Test data (4,000)	Classification data (10,000)
no normalization	-11.4	108.6
min-max normalization	38.1	-22.8
Z normalization	26.7	19.2
decimal scaling	43.8	120.0

Since we do not to achieve acceptable results here either, we refer back to the considerations we made in the experiments on k-nearest neighbor and naive Bayes. We repeat the experiments with the following configuration:

- Decision tree; only the three features *payment_type*, *power_consumption*, and *HHH*; *pc*, and *HHH* are binned; -1 for missing values; equal size sampling; KNIME.

This significantly improves the situation. If we group the features *HHH* (0..11) and *power_consumption* (0..5), we obtain between € 3,000 and € 5,000. If we group the features in the second variant, *power_consumption* (0..3) and *HHH* (0..6), the result increases up to € 6,000.

As a reminder, the tests will yield different results because although the training data is always divided using stratified sampling, the selection is still random. What is striking about the tests conducted so far is that this random factor usually causes a large variance of results.

The following table lists the best results achieved with a decision tree:

Result	Characterization of the experiment
6,665.70	J 48, only 3 attributes
	missing values: -1
	coding of <i>power_consumption</i> (discrete): 0..5
	coding of <i>HHH</i> (discrete): 0..11
	equal size sampling
6,300.30	as before
	coding of <i>power_consumption</i> (discrete): 0..3
	coding of <i>HHH</i> (discrete): 0..6

Here, too, we observe that the result is highly dependent on the generation of the training set. Therefore, we will experiment again with generating our own

test set. The results are slightly worse, but show only a small variance. The best decision tree based on the training data achieves € 6116.10.

Result	Characterization of the experiment
6,116.10	J 48, only 3 attributes
	missing values: -1
	coding of <i>power_consumption</i> (discrete): 0..3
	coding of <i>HHH</i> (discrete): 0..6
	separate training set: 800 cancelers, 1000 non-cancelers

We achieve acceptable results with the methods used to date. However, this does require a some “deep thinking” about the data. Simply applying the methods is rarely successful.

We note that it can be useful to establish comparability of the number of data records with regard to the target attribute. We also note that the results of some tests show a significant variance. Only a separate concept for splitting into training and test sets produces stable and good results.

10.5.4 Neural networks

We conduct most of our experiments with artificial neural networks. In addition to KNIME, we also use JAVANNNS for this purpose. In neural networks, both the network architecture and the training influence the behavior of the network. We use a feedforward-neural network, a *MultiLayer Perceptron* (MLP). The architecture of the network for a classification task is largely generated automatically in KNIME:

- The input layer contains exactly one neuron for each feature. For the example, the network thus contains 32 input neurons.
- The number and size of the hidden layers can be adjusted via the configuration menu. By default, an inner layer with 10 neurons is assumed, which is a reasonable size for our example with its 32 input neurons.
- The size of the output layer is determined by the classification attribute: one neuron is used for each value of the feature. For the example, two output neurons are used, one for the class *yes* and one for the class *no* of the feature *canceler*. The class is then predicted for a data record whose neuron has the highest activation.

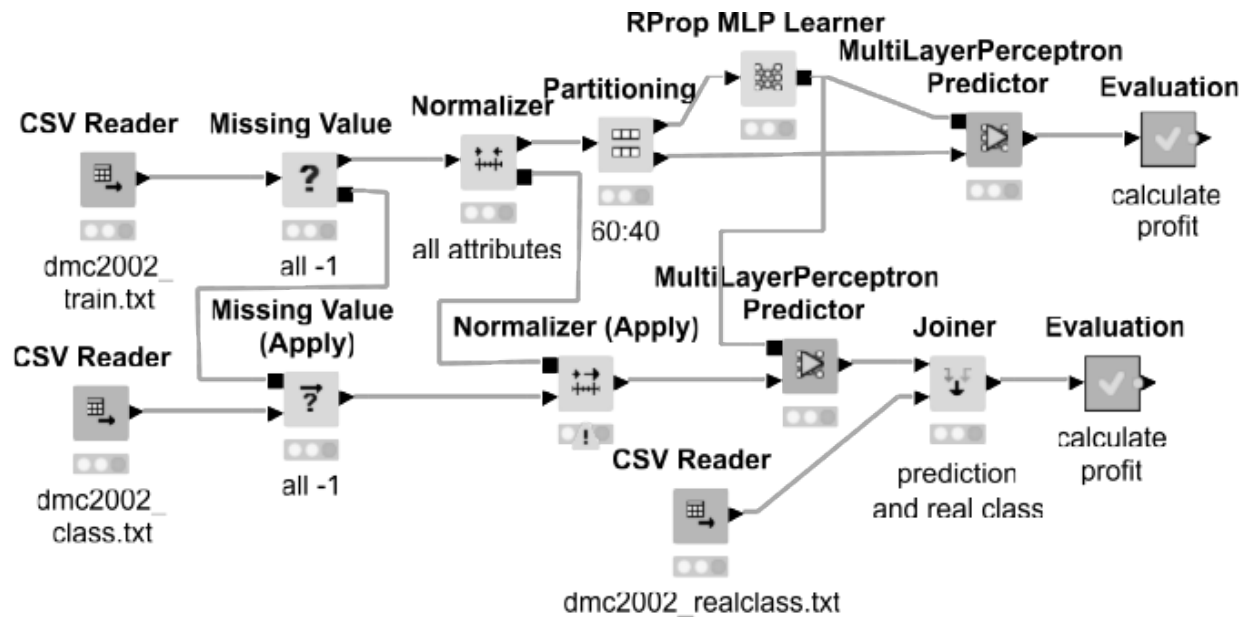


Fig. 10.9 Experiments with a neural network

Experiment 7 (→ Figure 10.9) uses all features. The attributes *power_consumption* and *HHH* are not encoded, and -1 is used for all missing values. Decimal scaling is used. The procedure is similar again, only the nodes for working with neural networks, the *MLP learner* and the *MLP predictor*, are used. We experiment with two different inner layers consisting of 10 and 20 neurons, respectively. More than one inner layer does not seem appropriate due to the size of the network. The network will be trained for different lengths of time. Longer training can improve the result, but training that is too long leads to *over-fitting*: the training data is processed better, but the error rate for the test data increases. This effect can also be seen in the results tables. Thus, in → Tables 10.9 and → 10.10, the result after 3,000 learning cycles is smaller than after 1,000 learning cycles.

Tab. 10.9 Results of experiment 7, min-max normalized data

Hidden layer	Learning cycles	Test data	Classification data
1x10	100	221.1	217.5
	300	413.7	657.6
	500	594.9	387.0
	1000	704.1	1039.2
	3000	483.6	994.2
1x20	100	184.8	232.5
	300	236.7	619.8
	500	250.5	618.3
	1000	397.8	897.6
	3000	899.4	1563.3
	4000	975.6	1654.8
	6000	813.6	1277.7

Tab. 10.10 Results of experiment 7, decimal scaled data

Hidden layer	Learning cycles	Test data	Classification data
1 x 10	100	0.0	0.0
	300	143.1	393.0
	500	380.1	543.6
	1000	303.9	1569.3
	3000	428.4	922.5
1 x 20	100	38.1	-11.4
	300	457.8	1686.3
	500	271.2	1867.8
	1000	131.7	1850.1
	3000	756.6	120.8

The same experiments are now performed with the coded values for *power_consumption* and *HHH*. With the encoded data, the respective median for missing values, and decimal scaling the MLP network achieves the results listed in → Table 10.11. Surprisingly, grouping the values for *power_consumption* and truncating the high values for *HHH* does not lead to an improvement.

We will now use `JAVANNNS` to build and train a neural network for the same data. Creating a neural network in `JAVANNNS` is described in → Section 1.6.3. The neural network again consists of 32 input neurons, one neuron for each

feature, and, in this case, only one output neuron. If this is activated, the class *yes* is predicted; if it is not activated, the class *no* is predicted. We define an hidden layer of 16 neurons. → Figure 10.10 shows the network and the test environment with the *control window* and the network *error display*.

Tab. 10.11 Results of experiment 7 with grouped values for *pc* and *HHH*

Hidden layer	Learning cycles	Test data	Classification data
1 x 10	100	0.0	0.0
	300	423.1	120.6
	500	73.2	392.4
	1000	527.7	930.9
	3000	885.9	1654.2
1x20	100	0.0	-11.4
	300	404.4	584.1
	500	690.3	978.9
	1000	548.4	1265.1
	3000	927.7	1282.8

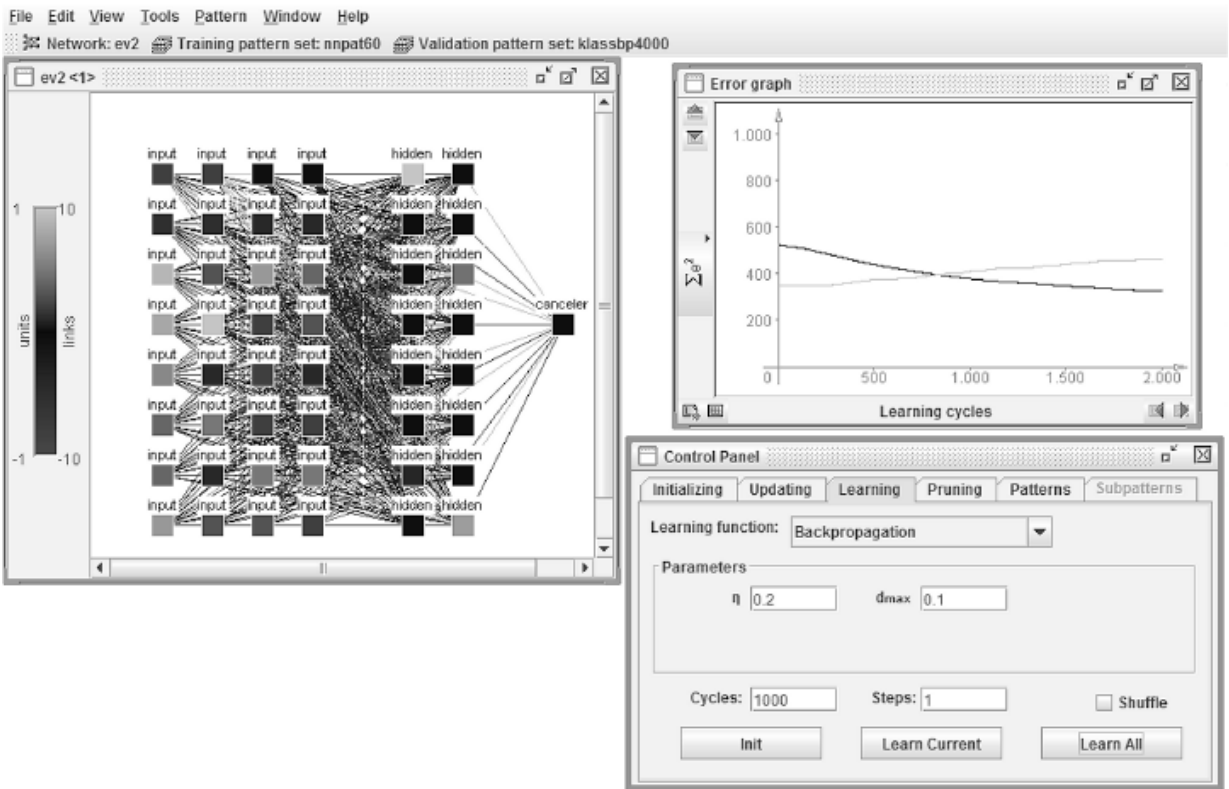


Fig. 10.10 Neural network for Experiment 8 in JavaNNS

When using JAVANNNS, a corresponding program is required to evaluate the network prediction and calculate the objective function. The network output is available as a *res* file, which lists the expected output and the calculated output for all inputs. This *res* file must be evaluated.

```
SNNS result file V1.4-3D
generated at Mon Aug 11 10:13:35 2025
No. of patterns      : 10000
No. of input units  : 32
No. of output units : 1
startpattern : 1
endpattern   : 10000
teaching output included
#1.1
0
0.12242
#2.1
0
0.11077
#3.1
...
#102.1
1
0.15244
...
```

The authors use a small Java program to evaluate the *res* file, which displays the number of correctly identified classes and the result for the objective function.

Tab. 10.12 Results of experiment 8, decimal scaled data

Learning parameter	Learning cycles	Test data	Classification data
BackProp 0.2	100	0	0
	300	20	209
BackProp 0.5	100	0	43
RPROP	100	26	64
	300	316	477

The results in → Table 10.12 are rather disappointing. The table shows that no or very few cancelers are identified; few ones are detected by the network. The *res* file shows that all values calculated by the network are very small and values close to one do not occur at all or hardly ever. However, a simple idea helps the network to achieve high performance: instead of choosing the threshold value of 0.5 as the boundary between 1 and 0, the classes *yes* and *no*, we determine the most favorable threshold value. In terms of neural networks, the output of the neurons is calculated from the activation using an *output function*. In practice, the output *out* is determined as follows:

```
out = (int) Math.round(act+shift);
```

A value *shift* is added to the calculated activation *act* to make it easier to detect a 1. This calculation is integrated into the evaluation program so that a favorable *shift* can be easily determined. Correctly, we determine the *shift* based on the test set, which consists of 40 % of the data records from the original training set. This *shift* is then applied to the entire classification data, and we obtain a result that can be compared with the competition results.

→ Table 10.13 shows the impressive improvements that can be achieved with this *shift* in the threshold value of the output neuron. This approach has been successfully applied in several projects.

Tab. 10.13 Results of experiment 8 with modified threshold

Learning parameter	Learning cycles	Bias shift	Test data	Classification data
BackProp 0.2	100	0.37	2048	4243
	300	0.34	1383	3081
BackProp 0.5	100	0.38	1571	5076
RPROP	100	0.35	2803	6573
	300	0.34	1871	4865

Once again, we conduct largely the same experiment, except that this time the values of the features *power_consumption* and *HHH* are binned as described in → Section 10.4. Here, we directly provide the table with the bias *shift*, as only these experiments achieve good results, see → Table 10.14.

Tab. 10.14 Results of experiment 8 with encoded values for *pc* and *HHH*

Learning parameter	Learning cycles	Bias shift	Test data	Classification data
BackProp 0.2	50	0.34	3130	5741
	100	0.33	3223	7133
	300	0.33	3017	4930
BackProp 0.5	100	0.33	2984	6265
RPROP	100	0.33	2965	5520
	300	0.33	2486	4370

To conclude our experiments, we also try the variant with only the three features *payment_type*, *power_consumption*, and *HHH* using neural networks. Experiment 9 uses KNIME for this purpose, and Experiment 10 uses JAVANNNS. The missing values are replaced by the median of the respective column, and all data is scaled to the range 0..1. As in the corresponding experiment for decision tree learning, a column selection node is inserted between the encoding of the features and the replacement of the missing values. → Figure 10.11 shows the data preprocessing. All further steps are identical to the procedure in → Figure 10.9.

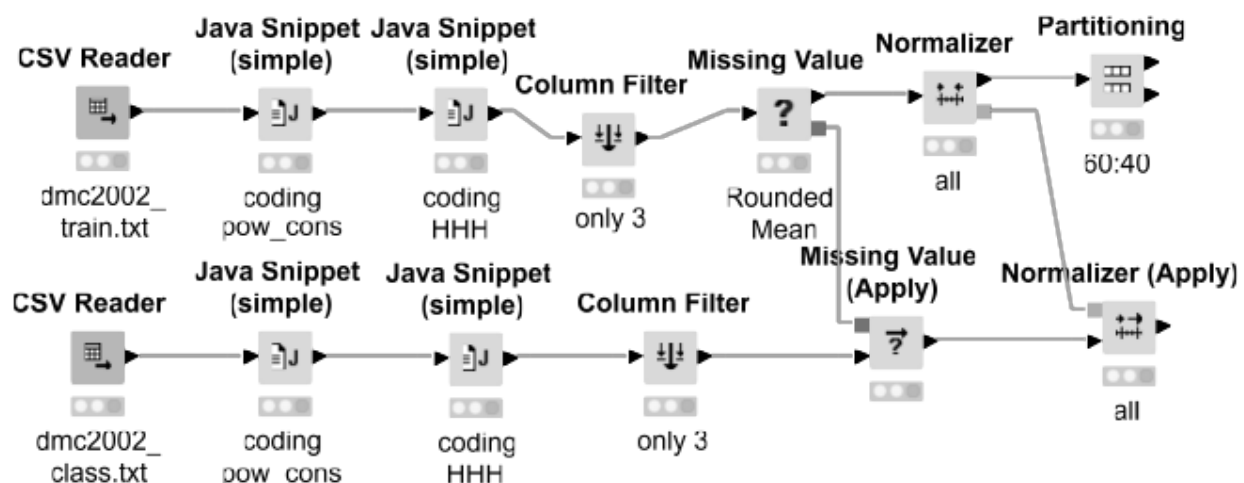


Fig. 10.11 Preprocessing for experiments 9 and 10

This experimental setup does not yield good results: with an hidden layer of 10 neurons, even after 1,000 learning cycles, the objective function does not produce a result greater than zero. If 20 neurons are used in the hidden layer,

a result of € 43.80 for the test data and € 120.0 for the classification data is only achieved after 1,000 learning cycles.

Despite these poor results, we will carry out Experiment 10, i.e., train a network in JAVANNS with the same data. → Figure 10.12 shows the architecture and the learning environment.

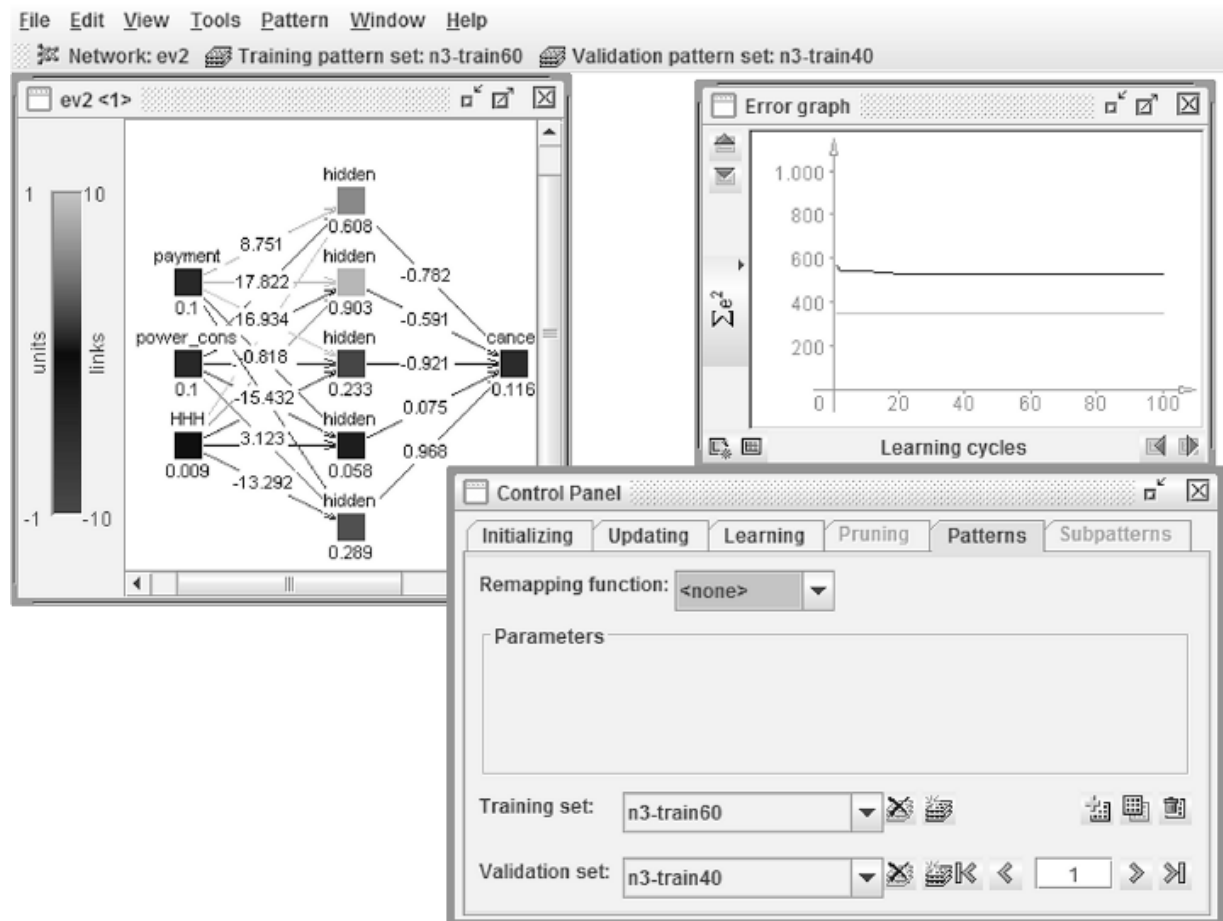


Fig. 10.12 A neural network with only three inputs

Training is carried out using the training set, namely the pattern file *tr60.pat*, while the pattern file *tr40.pat* is used for testing. The *error graph* window shows the error curve for the training set (upper line) and the error for the test set (lower line). In the experiments with JAVANNS, the course of the error curves is tracked very closely. As the number of training cycles increases, the error for the training set decreases, but the error for the test set increases. Training must be stopped before this effect occurs.

Using the JAVANNNS network, we obtain the results listed in → Table 10.15 for our reduced input.

Tab. 10.15 Results of experiment 10, only three input features

Learning parameter	Learning cycles	Bias shift	Test data	Classification data
BackProp 0.1	100	0.35	2937	6198
	300	0.37	2951	6792
RPROP	100	0.37	0	6150

Without the trick of shifting the threshold value, no learning effect is achieved, which is why the table only shows the results using the lowered threshold value. At € 6,792.00, this experiment not only achieves a value that would have placed it seventh in the 2002 winners' list, but also achieves this with minimal input and a very small network.

10.6 Python program

The *Python* programming language is now the undisputed number one in the field of machine learning. In this section, we will use *Jupyter Notebook* and *Python*, along with its machine learning packages, to work through the example (see → Section 1.6.4).

If any packages are still missing, they can also be installed directly in Jupyter Notebook²:

```
# Install required packages
!pip install scikit-learn
!pip install seaborn
!pip install missingno
!pip install torch
!pip install tensorflow
```

The packages must be installed before they can be imported into the program. If errors are reported, the missing packages must be added as described above.

```
# Import necessary modules
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
```

10.6.1 Data preprocessing

If the imports have been completed without errors, the data can be read in and data preparation can begin.

The file *data_dmc2002_train_3attribute.csv* contains the training data with the attributes *ID*, *payment_type*, *power_consumption*, and *HHH*. The data is stored in the *Panda DataFrame* named *df*.

```
# Read the CSV file
%# delimiter ';' is used to handle csv in German Excel
df = pd.read_csv('data_dmc2002_train_3attribute.csv',
delimeter=';')
print(type(df))
```

```
<class 'pandas.core.frame.DataFrame'>
<bound method NDFrame.head of
```

	ID	payment_type	power_consumption	HHH	canceler
0	22891	2	138	7.0	no
1	26663	1	6034	2.0	no
2	2253	2	4996	2.0	no
3	26769	2	1514	1.0	no
4	11264	2	3115	8.0	no
...	...	...	...	...	...
9995	18413	1	2053	1.0	no
9996	1853	2	1504	1.0	no
9997	34366	1	1765	2.0	no
9998	7695	2	4108	1.0	yes

9999	19769	1	1267	1.0	no

[10000 rows x 5 columns]>

Now the data preprocessing can begin. First, missing values are filled in. For this purpose, the package provides the *fillna* function, which replaces *NaN* values (not a number) with a numeric value. Here, the median of the attribute in the data set is used.

```
# Fill missing values in numeric columns with the median
value
numeric_columns      = df.select_dtypes(include=[float,
int]).columns
df[numeric_columns] = df[numeric_columns].\
                        fillna(df[numeric_columns].median())
```

As in the other experiments, the three input attributes are preprocessed. First, we look at *HHH*, and then *power_consumption*, in order to handle the outliers here. For *HHH*, the frequencies of the occurring values are determined, and for all frequencies smaller than a threshold value, the code 15 is assigned to the corresponding attribute value.

```
# Preprocessing 'HHH'

# Get the frequency counts of each value in 'HHH'
value_counts = df['HHH'].value_counts()

# Identify the values below a threshold
# and assign '15' to all values with frequency < threshold
threshold = 10
below_threshold_values = \
    value_counts[value_counts < threshold].index.tolist()
df.loc[df['HHH'].isin(below_threshold_values), 'HHH'] = 15

# Convert the 'HHH' column to the integer data type
```

```
df['HHH'] = df['HHH'].astype(int)

# show the frequency distribution of 'HHH'
counts = df['HHH'].value_counts()
counts.plot(kind='bar')
plt.show()
print(counts)
```

The last paragraph of the box serves solely to output the distribution, see
→ Figure 10.13.

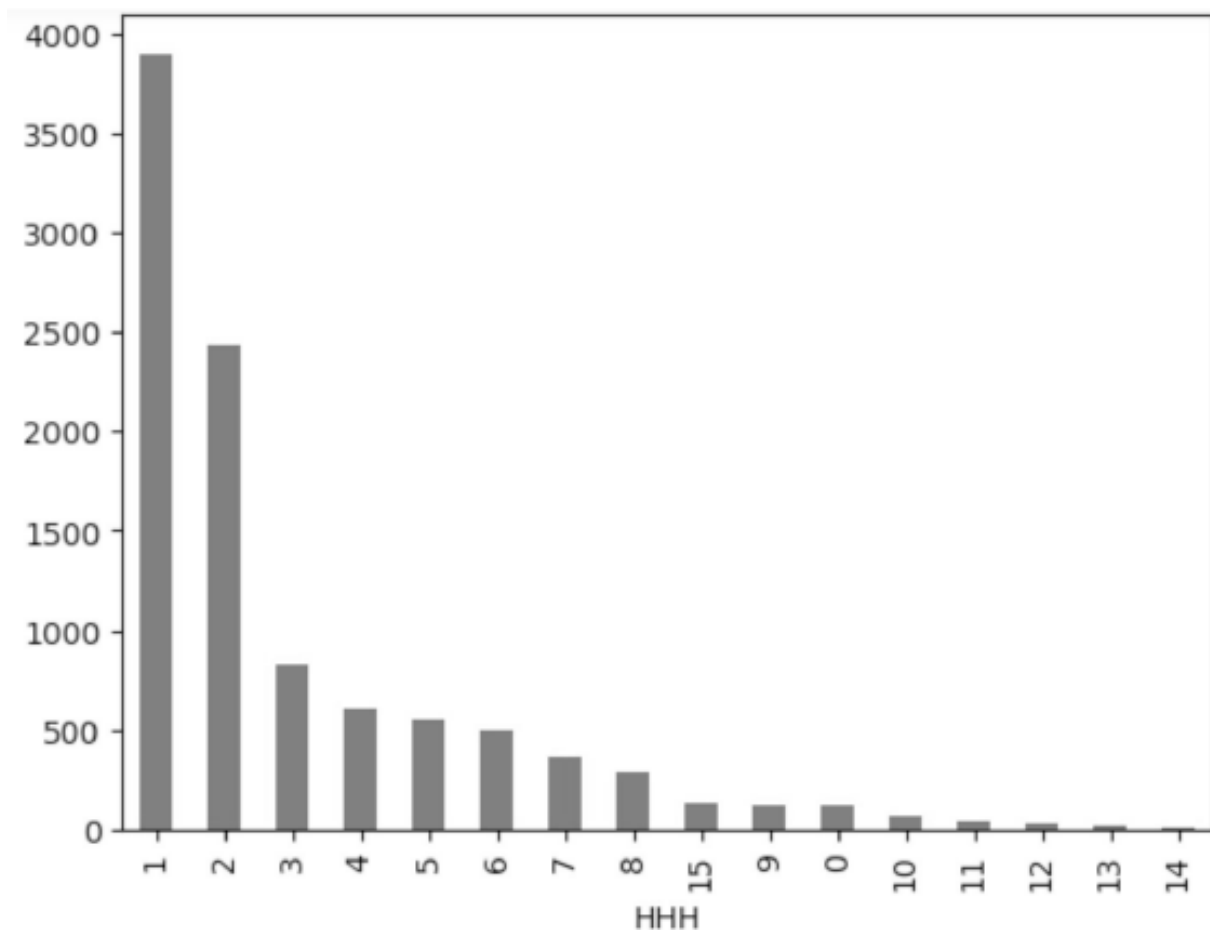


Fig. 10.13 Frequency distribution of the attribute *HHH*

```
HHH
1      3896
```

2	2437
3	825
4	610
5	552
6	496
7	360
8	282
15	131
9	122
0	117
10	65
11	42
12	28
13	26
14	11

Name: count, dtype: int64

The values are now normalized. To do this, a min-max normalizer is created, which can then be used step by step for all three attribute columns. First, the *HHH* values are normalized.

```
# MinMaxScaler object for normalization of
# 'HHH', 'power_consumption' and 'payment_type'
scaler = MinMaxScaler()

# Normalize 'HHH'
df['HHH'] = scaler.fit_transform(df[['HHH']])
print(df['HHH'])
```

0	0.466667
1	0.133333
2	0.133333
3	0.066667
4	0.533333
	...
9995	0.066667

```
9996    0.066667
9997    0.133333
9998    0.066667
9999    0.066667
Name: HHH, Length: 10000, dtype: float64
```

We proceed similarly with the attribute *power_consumption*, and then we process the values of the attribute *payment_type*. The value 3 occurs extremely rarely, so it is considered an outlier and replaced by the value 2.

```
# Preprocessing 'power_consumption'

# Replace values higher than 9999 by 10,000 and make it
integer
df.loc[df['power_consumption'] > 9999, 'power_consumption'] =
10000
df['power_consumption'] = df['power_consumption'].astype(int)

# show the frequency distribution of power_consumption
counts = df['power_consumption'].value_counts()
print(counts)
```

```
power_consumption
0          182
10000       64
1538        46
1492        32
1          29
...
223         1
7222        1
2823        1
3206        1
3117        1
Name: count, Length: 4575, dtype: int64
```

```
# Normalize 'power_consumption'
df['power_consumption'] = \
    scaler.fit_transform(df[['power_consumption']])
print(df['power_consumption'])
```

```
0      0.0138
1      0.6034
2      0.4996
3      0.1514
4      0.3115
```

...

```
9995    0.2053
9996    0.1504
9997    0.1765
9998    0.4108
9999    0.1267
```

Name: power_consumption, Length: 10000, dtype: float64

```
# Preprocessing 'payment_type'
# Get the frequency distribution of 'payment_type'
payment_type_freq = df['payment_type'].value_counts()
print(payment_type_freq)
# Replace the few '3' by '2'
df['payment_type'].replace(3, 2, inplace=True)
# normalize 'payment_type'
df['payment_type'] = scaler.fit_transform(df[['payment_type']])
```

payment_type

```
2      7329
1      2649
3         22
```

Name: count, dtype: int64

```
# Print data type of each feature
for column in df.columns:
    print(f>Data type of {column}: {df[column].dtype}>)
# display the normalized
data print(df.head())
```

At the end of preprocessing, the encoded data is displayed, allowing the user to get a final impression of the preprocessing. The result of the preprocessing is then written to a csv file *preprocessed_train_dataset.csv*.

```
Data type of ID: int64
Data type of payment_type: float64
Data type of power_consumption: float64
Data type of HHH: float64
Data type of canceler: object
```

	ID	payment_type	power_consumption	HHH	canceler
0	22891	1.0	0.0138	0.466667	no
1	26663	0.0	0.6034	0.133333	no
2	2253	1.0	0.4996	0.133333	no
3	26769	1.0	0.1514	0.066667	no
4	11264	1.0	0.3115	0.533333	no

```
# Look at the distribution of the canceler values
class_counts = df['canceler'].value_counts()

print("Class Distribution:")
print(class_counts)
```

```
Class Distribution:
canceler
no      9000
yes     1000
Name: count, dtype: int64
```

```
# Save the preprocessed data to a new CSV file
df.to_csv("preprocessed_train_dataset.csv", sep=';',
index=False)
```

10.6.2 Model development

For model development using *Python*, we choose to build and train a neural network. The architecture of a neural network is characterized by the number of its (trainable) layers of neurons and their interconnections. The structure of the input layer is determined by the number of available attributes, which is 3. A single neuron is used as the output, which, when activated, indicates a canceler (*yes*). A zero means that it is a non-canceler (*no*).

Model development consists of the following steps:

1. Import the required packages or parts thereof: *pandas*, *sklearn*, *tensorflow*.
2. Read the preprocessed data from the csv file *preprocessed_train_dataset.csv*, then output some of the data.
3. Divide the training data into a set for training and a set for testing the model.
4. Build the neural network: specify the number and size of the hidden layers as well as the activation function used.
5. Train the network, whereby different numbers of training cycles should be tried.
6. Evaluate the model in two steps: First, for the test data, convert the activation of the output neuron to 0 or 1 using a threshold value. Then compare this output with the desired output and calculate the *accuracy* of the model's prediction.
7. Finally, save the developed model in a file so that it is now available for use.

The steps listed above can be easily found in the following source code sections. At points where the user can influence the experiments, we will discuss some experiences and provide hints.

```
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.preprocessing import LabelEncoder
```

```
# Read the preprocessed data from a csv file
df = pd.read_csv('preprocessed_train_dataset.csv',
delimiter=';')
```

```
# Display the training data
print(df.head())
for column in df.columns:
    print(f>Data type of {column}: {df[column].dtype}")
```

	ID	payment_type	power_consumption	HHH	canceler
0	22891	1.0	0.0138	0.466667	no
1	26663	0.0	0.6034	0.133333	no
2	2253	1.0	0.4996	0.133333	no
3	26769	1.0	0.1514	0.066667	no
4	11264	1.0	0.3115	0.533333	no

```
Data type of ID: int64
Data type of payment_type: float64
Data type of power_consumption: float64
Data type of HHH: float64
Data type of canceler: object
```

```
# Separate the features (X) and the target variable (y)
X = df[['payment_type', 'power_consumption', 'HHH']]
y = df['canceler']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = \
```



```

train_test_split(X, y, \
                  test_size=0.2, \
                  random_state=42)

# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Encode the target column
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

class_counts = y_train.value_counts()
print('yes-no distribution in test set:', class_counts)

```

```

yes-no distribution in test set: canceler
no          7229
yes         771
Name: count, dtype: int64

```

The user has the option of designing the size of the network by adding hidden layers using the add function. The number of neurons in these layers influences the performance of the network. Too many neurons can lead to overfitting, meaning that the network learns the training data too well but is not good at assigning the data to be classified. Experiment with different hidden layers by changing the number of neurons in the layers. The second layer should not be larger than the first. The activation function can also be easily varied; instead of *'sigmoid'*, you can also try *'relu'*.

```

# parameter of the neural network
hidden_neurons_1 = 30
hidden_neurons_2 = 15
act_fct          = 'sigmoid'
training_cycles  = 50

# create the artificial neural network
model = keras.Sequential()

```

```
# add hidden layer and an output layer
model.add(layers.Dense(hidden_neurons_1, \
                        activation=act_fct, input_shape=(3,)))
model.add(layers.Dense(hidden_neurons_2, \
                        activation=act_fct))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', \
              optimizer='adam', metrics=['accuracy'])
```

```
# Train the model
model.fit(X_train, y_train_encoded, \
          epochs=training_cycles, batch_size=64)
```

```
Epoch 1/50
125/125 [=====] - 1s 2ms/step - loss:
0.4237
- accuracy: 0.8586
Epoch 2/50
125/125 [=====] - 0s 2ms/step - loss:
0.3177
- accuracy: 0.9036
Epoch 3/50
125/125 [=====] - 0s 2ms/step - loss:
0.3160
- accuracy: 0.9036
...
Epoch 49/50
125/125 [=====] - 0s 3ms/step - loss:
0.3050
- accuracy: 0.9036
Epoch 50/50
125/125 [=====] - 0s 2ms/step - loss:
0.3049
- accuracy: 0.9036
```

```
# Evaluate the model
y_pred_prob = model.predict(X_test)
print(y_pred_prob)
y_pred = (y_pred_prob > 0.2).astype(int).flatten()
print(y_pred)
```

```
63/63 [=====] - 0s 3ms/step
[[0.05598645]
 [0.19949497]
 [0.07314616]
 ...
 [0.05175177]
 [0.09473965]
 [0.10602566]]
[0 0 0 ... 0 0 0]
```

```
# what is the accuracy of the model?
accuracy = (y_pred == y_test_encoded).mean()
print('Test Accuracy:', accuracy)
```

Test Accuracy: 0.8635

```
# save the model
model.save('trained_model.h5')
print("Model saved successfully.")
```

Model saved successfully.

10.6.3 Application of the model for a prediction

The data to be classified is preprocessed in a separate step. The model accesses this data and classifies it as *cancelers* or *non-cancelers*.

The prediction is then compared with the actual values and the objective function is calculated.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
```

```
# Read the CSV file
df = pd.read_csv('data_dmc2002_class_3attribute.csv',
delimeter=';')
print(df)
```

```
# Fill missing values with the median
numeric_columns = df.select_dtypes(include=[float,
int]).columns
df[numeric_columns] = \

df[numeric_columns].fillna(df[numeric_columns].median())

# Define the threshold for categorization
threshold = 10

# Get the frequency counts of each value in 'HHH'
value_counts = df['HHH'].value_counts()
below_threshold_values = \
    value_counts[value_counts < threshold].index.tolist()

# Assign the values below the threshold to the category '15'
df.loc[df['HHH'].isin(below_threshold_values), 'HHH'] = 15
df['HHH'] = df['HHH'].astype(int)

# Calculate the frequency distribution of the column HHH
counts = df['HHH'].value_counts()
```

```
counts.plot(kind='bar')  
plt.show()  
print(counts)
```

```
# Create a MinMaxScaler object  
scaler = MinMaxScaler()  
  
# Normalize the 'HHH' column  
df['HHH'] = scaler.fit_transform(df[['HHH']])  
  
# Print the updated 'HHH' column values  
print(df['HHH'])
```

```
# Replace values above 9999 with 10000 and make it int  
df.loc[df['power_consumption'] > 9999, 'power_consumption'] =  
10000  
df['power_consumption'] = df['power_consumption'].astype(int)
```

```
# Normalization of 'HHH', 'power_consumption' and  
'payment_type'  
# Create a MinMaxScaler object  
scaler = MinMaxScaler()  
  
# Normalize the 'power_consumption' column  
df['power_consumption'] = \  
scaler.fit_transform(df[['power_consumption']])  
print(df['power_consumption'])
```

```
# Get the frequency distribution of 'payment_type'  
payment_type_freq = df['payment_type'].value_counts()  
print(payment_type_freq)
```

```
# Replace "3" with "2" in 'payment_type'
df['payment_type'].replace(3,2,inplace=True)
# normalize:
df['payment_type'] = scaler.fit_transform(df[['payment_type']])
```

```
# print the preprocessed data
for column in df.columns:
    print(f>Data type of {column}: {df[column].dtype}")
print(df.head())
```

Data type of ID: int64

Data type of payment_type: float64

Data type of power_consumption: float64

Data type of HHH: float64

ID	payment_type	power_consumption	HHH

0	28603	1.0	0.3748	0.133333
1	22558	1.0	0.0860	0.466667
2	37170	0.0	0.5705	0.066667
3	25059	0.0	0.1587	0.400000
4	33605	0.0	0.0329	0.066667

```
# Save the preprocessed data to a new CSV file
df.to_csv("preprocessed_class_dataset.csv",sep=';',
index=False)
```

Applying the model

It makes sense to handle the prediction in a separate *Jupyter notebook* file. In the experiments, you can then quickly switch from modifying the model to applying it. The model is applied in the following steps:

1. Import the required packages

2. Read in the preprocessed data and output it for illustration purposes
3. Read in the model
4. Classify the data, applying the model
5. Follow-up on the results
6. Comparison with actual results and calculation of the objective function

```
import pandas as pd
from tensorflow import keras
from sklearn.preprocessing import LabelEncoder
```

```
# Read the CSV file
df =
pd.read_csv('preprocessed_class_dataset.csv', delimiter=';')
print(df)
```

	ID	payment_type	power_consumption	HHH
0	28603	1.0	0.3748	0.133333
1	22558	1.0	0.0860	0.466667
2	37170	0.0	0.5705	0.066667
3	25059	0.0	0.1587	0.400000
4	33605	0.0	0.0329	0.066667
...	...	...	...	...
9995	954	1.0	0.4577	0.133333
9996	28928	1.0	0.5151	0.066667
9997	33602	0.0	0.2309	0.200000
9998	30740	0.0	0.2679	0.266667
9999	25610	1.0	0.2539	0.066667
[10000 rows x 4 columns]				

```
# Load the model
loaded_model = keras.models.load_model('trained_model.h5')
print("Model loaded successfully.")
```

Model loaded successfully.

```
# Split the data into features (X) and target (y)
X_pred = df[['payment_type', 'power_consumption', 'HHH']]

# Make predictions
y_pred_prob = loaded_model.predict(X_pred)
# bias lower than 0.5 improves the result:
y_pred = (y_pred_prob > 0.5).astype(int).flatten()

# Decode the predicted labels
label_encoder = LabelEncoder()
label_encoder.fit_transform(['no', 'yes'])

# Convert labels back to their original values
y_pred_labels = label_encoder.inverse_transform(y_pred)

# Add the predicted labels to the DataFrame and show it
df['canceler_prediction'] = y_pred_labels
print(df.head())
```

313/313 [=====] - 0s 1ms/step

	ID	payment_type	power_consumption	HHH	canceler_prediction
0	28603	1.0	0.3748	0.133333	no
1	22558	1.0	0.0860	0.466667	no
2	37170	0.0	0.5705	0.066667	yes
3	25059	0.0	0.1587	0.400000	no
4	33605	0.0	0.0329	0.066667	no

```
# Save ID column and prediction to a text file
pred= pd.concat([df['ID'],df['canceler_prediction']], axis=1)
pred.to_csv('prediction.txt', index=False)

print(pred.head())
```



```
# Calculate the percentage of "yes" and "no" predictions
percentage_yes = (y_pred == 1).mean() * 100

percentage_no = (y_pred == 0).mean() * 100

print(f"Percentage of 'yes' predictions:
{percentage_yes:.2f}%")
print(f"Percentage of 'no' predictions:
{percentage_no:.2f}%")
```

	ID	canceler_prediction
0	28603	no
1	22558	no
2	37170	yes
3	25059	no
4	33605	no

Percentage of 'yes' predictions: 26.88 %
Percentage of 'no' predictions: 73.12 %

```
# Compare the prediction and the real class
# Read realclass and create confusion matrix
realclass = pd.read_csv('data_dmc2002_realclass.csv')

from sklearn.metrics import confusion_matrix, make_scorer
realx = realclass['canceler']
predx = df['canceler_prediction']
print(confusion_matrix(realx, predx))
falseNeg, falsePos, trueNeg, truePos = \
    confusion_matrix(realx, predx).ravel()
# Goal function
print(truePos*43.80 - falsePos*5.70)
```

[[6743 2257]

[569 431]]
6012.9

We achieved results between 6,000 and 7,000 with the Python program. For this, we tried hidden layers with up to a size of 64 neurons in both layers and both activation functions. As already known from experiments with neural networks using JAVANNNS, classification as a canceler is difficult to achieve. Technically speaking, the network finds it difficult to learn an activated output neuron with an activity greater than 0.5. With a smaller threshold value, say between 0.1 and 0.2, however, the results improve dramatically and lead to the values mentioned above. For example, a significantly smaller network with only 20 neurons in the first and 5 neurons in the second hidden layer yields a result of 6,782.10 after 50 training cycles. There is plenty of scope for experimentation. Make use of it.

10.7 Evaluation of the results

The experiments carried out cannot be considered complete for the processing of a realistic data mining task. On the one hand, all data mining algorithms are mathematical and therefore exact procedures. On the other hand, only a large number of experiments can achieve a desired or at least useful result. There is currently no model that shows us a successful path to the result depending on a task. This experimentation is then the only alternative to a purely mathematical approach, which is usually not possible for practical problems.

However, our experiments have shown that the energy supplier can generate significant additional revenue by using a prediction model for its discount campaign. The experiments with the best results are listed in → Table 10.16.

Tab. 10.16 The best results of our experiments

Result	Characterization of the experiment
7,133	neural network, all features, <i>power_consumption</i> and <i>HHH</i> binned, decimal scaled, JAVANNS
6,792	neural network, only 3 features (<i>payment_type</i> , <i>power_consumption</i> , and <i>HHH</i>), <i>power_consumption</i> and <i>HHH</i> binned, decimal scaled, JAVANNS
6,573	neural network, all features, no binning, decimal scaled, JAVANNS
6,533.10	k-nearest neighbor, k=7, only 3 attributes, decimal scaled, no weighted distance, missing values: -1, coding of <i>power_consumption</i> (discrete): 0..3, coding of <i>HHH</i> (discrete): 0..6, separate training set: 800 cancelers, 1000 non-cancelers, KNIME
6,355.50	naive Bayes, only 3 attributes, missing values: -1, coding of <i>power_consumption</i> (discrete): 0..5, coding of <i>HHH</i> (discrete): 0..11, equal size sampling, KNIME
6297.90	as before, coding of <i>power_consumption</i> (discrete): 0..3, coding of <i>HHH</i> (discrete): 0..6, KNIME
6,165	naive Bayes, only 3 attributes, missing values: -1, coding of <i>power_consumption</i> (discrete): 0..3, coding of <i>HHH</i> (discrete): 0..6, separate training set: 800 cancelers, 1000 non-cancelers, KNIME
6,665.70	J48, only 3 attributes, missing values: -1, coding of <i>power_consumption</i> (discrete): 0..5, coding of <i>HHH</i> (discrete): 0..11, equal size sampling, KNIME
6,300.30	as before, coding of <i>power_consumption</i> (discrete): 0..3, coding of <i>HHH</i> (discrete): 0..6, KNIME
6,116.10	J48, only 3 attributes, missing values: -1, coding of <i>power_consumption</i> (discrete): 0..3, coding of <i>HHH</i> (discrete): 0..6, separate training set: 800 cancelers, 1000 non-cancelers, KNIME

It is striking that the neural networks developed in JAVANNS show the best results. However, it should be noted that these results can only be achieved through creative analysis and further processing of the network outputs. Working with JAVANNS also requires the use of additional software for data preparation and for evaluating the network outputs. Experiments with other classifiers show that it is certainly possible to reach the order of magnitude of € 6,000.

What conclusions can be drawn from the experiments for the task from the Data Mining Cup 2002, the development of a prediction model for customers of an energy supplier?

- Several (many) experiments are always necessary.
- Various methods and different software tools should be used and tested.
- Prefabricated components can be used to quickly carry out data mining experiments. However, this does not replace an understanding of the methods used.
- A creative approach and the implementation of your own processing steps can lead to better results.

- Data preparation has a major impact on the results.
- Experimentation and the logging of these experiments are essential for successful data mining.
- If possible, experiments should be automated in order to try out many variants and thus increase the chance of good results. The loops presented in → Section 1.6.1, for example, are suitable for this purpose, as they allow a large number of variants to be processed in *asingle* run. KNIME offers a range of different loops.

The approaches described in the previous sections leave plenty of room for further experimentation. Here are a few possibilities:

- The distribution of the training set can be modified. Instead of 60:40, you can also try 70:30 or 80:20. It is important to ensure that the distribution of the classification feature is the same in the subsets. However, if the classes are very unevenly distributed, as in our example, then a different distribution of the data should be considered.
We have already discussed the uneven distribution of cancelers/non-cancelers and resolved it using the *Equal Size Sampling* node. However, this node has the disadvantage of drastically reducing the number of non-cancelers in order to achieve a 1:1 ratio. Alternatively, the number of cancelers can be multiplied. The goal is to increase the proportion of cancelers, not necessarily to achieve a 1:1 ratio. This approach is discussed in → Section 8.2.4. → Figure 8.5 shows an example of quadrupling the number of cancelers. However, the multiplication is only necessary on the training set, not on the test set.
- The grouping and encoding for the *power_consumption* feature can be modified. For example, instead of groups from 0 to 5, a finer subdivision of consumption values can be made: 11 bins for values up to 6,000 (0..10) and one bin for values greater than 6,000.
- For working with neural networks, all features can be binarized and corresponding networks developed. Here, too, one can try to limit the number of features to three.
- The architecture and learning parameters for training neural networks can be further varied.
- The binarization of features can also be used to develop association rules. Here, we are only interested in association rules that establish a connection with the classification feature.
If you decide to tackle this task, try to beat the result achieved by Tanja Ciernioch, a former student who scored a record € 7,771.80 in a semester

project in 2014. She achieved this using the naive Bayes method, and the key to her success was excellent, well-thought-out data preprocessing. We wish you every success and the necessary bit of luck in completing your data mining task.

A Appendix: Sample data

A.1 Iris data

One of the standard test data sets in data mining is the so-called Iris data set. This data is part of the sample data sets from WEKA. Edgar Anderson ¹collected data for the following Iris species:

- Iris setosa,
- Iris virginica,
- Iris versicolor.

The Iris data set consists of 50 observations of each of these three species, for which four attributes of the flowers were recorded, namely the length and width of the sepal and petal. The data therefore consists of the following attributes:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class Iris-setosa, Iris-versicolor, Iris-virginica

The data is available in the ARFF format, the WEKA standard format.

```
@RELATION Iris
@ATTRIBUTE sepallength REAL
@ATTRIBUTE sepalwidth REAL
@ATTRIBUTE petallength REAL
@ATTRIBUTE petalwidth REAL
@ATTRIBUTE class {Iris-setosa, Iris-versicolor, Iris-
virginica}
```

@DATA

5.1 3.5 1.4 0.2 Iris-setosa	7.0 3.2 4.7 1.4 Iris-versicolor	6.3
3.3 6.0 2.5 Iris-virginica		
4.9 3.0 1.4 0.2 Iris-setosa	6.4 3.2 4.5 1.5 Iris-versicolor	5.8
2.7 5.1 1.9 Iris-virginica		
4.7 3.2 1.3 0.2 Iris-setosa	6.9 3.1 4.9 1.5 Iris-versicolor	7.1
3.0 5.9 2.1 Iris-virginica		
4.6 3.1 1.5 0.2 Iris-setosa	5.5 2.3 4.0 1.3 Iris-versicolor	6.3
2.9 5.6 1.8 Iris-virginica		
5.0 3.6 1.4 0.2 Iris-setosa	6.5 2.8 4.6 1.5 Iris-versicolor	6.5
3.0 5.8 2.2 Iris-virginica		
5.4 3.9 1.7 0.4 Iris-setosa	5.7 2.8 4.5 1.3 Iris-versicolor	7.6
3.0 6.6 2.1 Iris-virginica		
4.6 3.4 1.4 0.3 Iris-setosa	6.3 3.3 4.7 1.6 Iris-versicolor	4.9
2.5 4.5 1.7 Iris-virginica		
5.0 3.4 1.5 0.2 Iris-setosa	4.9 2.4 3.3 1.0 Iris-versicolor	7.3
2.9 6.3 1.8 Iris-virginica		
4.4 2.9 1.4 0.2 Iris-setosa	6.6 2.9 4.6 1.3 Iris-versicolor	6.7
2.5 5.8 1.8 Iris-virginica		
4.9 3.1 1.5 0.1 Iris-setosa	5.2 2.7 3.9 1.4 Iris-versicolor	7.2
3.6 6.1 2.5 Iris-virginica		
5.4 3.7 1.5 0.2 Iris-setosa	5.0 2.0 3.5 1.0 Iris-versicolor	6.5
3.2 5.1 2.0 Iris-virginica		
4.8 3.4 1.6 0.2 Iris-setosa	5.9 3.0 4.2 1.5 Iris-versicolor	6.4
2.7 5.3 1.9 Iris-virginica		
4.8 3.0 1.4 0.1 Iris-setosa	6.0 2.2 4.0 1.0 Iris-versicolor	6.8
3.0 5.5 2.1 Iris-virginica		
4.3 3.0 1.1 0.1 Iris-setosa	6.1 2.9 4.7 1.4 Iris-versicolor	5.7
2.5 5.0 2.0 Iris-virginica		
5.8 4.0 1.2 0.2 Iris-setosa	5.6 2.9 3.6 1.3 Iris-versicolor	5.8
2.8 5.1 2.4 Iris-virginica		
5.7 4.4 1.5 0.4 Iris-setosa	6.7 3.1 4.4 1.4 Iris-versicolor	6.4
3.2 5.3 2.3 Iris-virginica		
5.4 3.9 1.3 0.4 Iris-setosa	5.6 3.0 4.5 1.5 Iris-versicolor	6.5
3.0 5.5 1.8 Iris-virginica		
5.1 3.5 1.4 0.3 Iris-setosa	5.8 2.7 4.1 1.0 Iris-versicolor	7.7
3.8 6.7 2.2 Iris-virginica		
5.7 3.8 1.7 0.3 Iris-setosa	6.2 2.2 4.5 1.5 Iris-versicolor	7.7
2.6 6.9 2.3 Iris-virginica		

5.1 3.8 1.5 0.3 Iris-setosa	5.6 2.5 3.9 1.1 Iris-versicolor	6.0
2.2 5.0 1.5 Iris-virginica		
5.4 3.4 1.7 0.2 Iris-setosa	5.9 3.2 4.8 1.8 Iris-versicolor	6.9
3.2 5.7 2.3 Iris-virginica		
5.1 3.7 1.5 0.4 Iris-setosa	6.1 2.8 4.0 1.3 Iris-versicolor	5.6
2.8 4.9 2.0 Iris-virginica		
4.6 3.6 1.0 0.2 Iris-setosa	6.3 2.5 4.9 1.5 Iris-versicolor	7.7
2.8 6.7 2.0 Iris-virginica		
5.1 3.3 1.7 0.5 Iris-setosa	6.1 2.8 4.7 1.2 Iris-versicolor	6.3
2.7 4.9 1.8 Iris-virginica		
4.8 3.4 1.9 0.2 Iris-setosa	6.4 2.9 4.3 1.3 Iris-versicolor	6.7
3.3 5.7 2.1 Iris-virginica		
5.0 3.0 1.6 0.2 Iris-setosa	6.6 3.0 4.4 1.4 Iris-versicolor	7.2
3.2 6.0 1.8 Iris-virginica		
5.0 3.4 1.6 0.4 Iris-setosa	6.8 2.8 4.8 1.4 Iris-versicolor	6.2
2.8 4.8 1.8 Iris-virginica		
5.2 3.5 1.5 0.2 Iris-setosa	6.7 3.0 5.0 1.7 Iris-versicolor	6.1
3.0 4.9 1.8 Iris-virginica		
5.2 3.4 1.4 0.2 Iris-setosa	6.0 2.9 4.5 1.5 Iris-versicolor	6.4
2.8 5.6 2.1 Iris-virginica		
4.7 3.2 1.6 0.2 Iris-setosa	5.7 2.6 3.5 1.0 Iris-versicolor	7.2
3.0 5.8 1.6 Iris-virginica		
4.8 3.1 1.6 0.2 Iris-setosa	5.5 2.4 3.8 1.1 Iris-versicolor	7.4
2.8 6.1 1.9 Iris-virginica		
5.4 3.4 1.5 0.4 Iris-setosa	5.5 2.4 3.7 1.0 Iris-versicolor	7.9
3.8 6.4 2.0 Iris-virginica		
5.2 4.1 1.5 0.1 Iris-setosa	5.8 2.7 3.9 1.2 Iris-versicolor	6.4
2.8 5.6 2.2 Iris-virginica		
5.5 4.2 1.4 0.2 Iris-setosa	6.0 2.7 5.1 1.6 Iris-versicolor	6.3
2.8 5.1 1.5 Iris-virginica		
4.9 3.1 1.5 0.1 Iris-setosa	5.4 3.0 4.5 1.5 Iris-versicolor	6.1
2.6 5.6 1.4 Iris-virginica		
5.0 3.2 1.2 0.2 Iris-setosa	6.0 3.4 4.5 1.6 Iris-versicolor	7.7
3.0 6.1 2.3 Iris-virginica		
5.5 3.5 1.3 0.2 Iris-setosa	6.7 3.1 4.7 1.5 Iris-versicolor	6.3
3.4 5.6 2.4 Iris-virginica		
4.9 3.1 1.5 0.1 Iris-setosa	6.3 2.3 4.4 1.3 Iris-versicolor	6.4
3.1 5.5 1.8 Iris-virginica		
4.4 3.0 1.3 0.2 Iris-setosa	5.6 3.0 4.1 1.3 Iris-versicolor	6.0
3.0 4.8 1.8 Iris-virginica		

5.1 3.4 1.5 0.2 Iris-setosa	5.5 2.5 4.0 1.3 Iris-versicolor	6.9
3.1 5.4 2.1 Iris-virginica		
5.0 3.5 1.3 0.3 Iris-setosa	5.5 2.6 4.4 1.2 Iris-versicolor	6.7
3.1 5.6 2.4 Iris-virginica		
4.5 2.3 1.3 0.3 Iris-setosa	6.1 3.0 4.6 1.4 Iris-versicolor	6.9
3.1 5.1 2.3 Iris-virginica		
4.4 3.2 1.3 0.2 Iris-setosa	5.8 2.6 4.0 1.2 Iris-versicolor	5.8
2.7 5.1 1.9 Iris-virginica		
5.0 3.5 1.6 0.6 Iris-setosa	5.0 2.3 3.3 1.0 Iris-versicolor	6.8
3.2 5.9 2.3 Iris-virginica		
5.1 3.8 1.9 0.4 Iris-setosa	5.6 2.7 4.2 1.3 Iris-versicolor	6.7
3.3 5.7 2.5 Iris-virginica		
4.8 3.0 1.4 0.3 Iris-setosa	5.7 3.0 4.2 1.2 Iris-versicolor	6.7
3.0 5.2 2.3 Iris-virginica		
5.1 3.8 1.6 0.2 Iris-setosa	5.7 2.9 4.2 1.3 Iris-versicolor	6.3
2.5 5.0 1.9 Iris-virginica		
4.6 3.2 1.4 0.2 Iris-setosa	6.2 2.9 4.3 1.3 Iris-versicolor	6.5
3.0 5.2 2.0 Iris-virginica		
5.3 3.7 1.5 0.2 Iris-setosa	5.1 2.5 3.0 1.1 Iris-versicolor	6.2
3.4 5.4 2.3 Iris-virginica		
5.0 3.3 1.4 0.2 Iris-setosa	5.7 2.8 4.1 1.3 Iris-versicolor	5.9
3.0 5.1 1.8 Iris-virginica		

A.2 Soybeans

The aim is to predict soybean diseases based on various factors, such as environmental influences. The data is included in WEKA as sample data. It originates from R. S. Michalski and R. L. Chilausky²

1. date: april, may, june, july, august, september, october
2. plant-stand: normal, lt-normal
3. precip: lt-norm, norm, gt-norm
4. temp: lt-norm, norm, gt-norm
5. hail: yes, no
6. crop-hist: diff-1st-year, same-1st-yr, same-1st-two-yrs, same-1st-sev-yrs
7. area-damaged: scattered, low-areas, upper-areas, whole-field

8.	severity:	minor, pot-severe, severe
9.	seed-tmt:	none, fungicide, other
10.	germination:	'90-100%', '80-89%', 'lt-80%'
11.	plant-growth:	norm, abnorm.
12.	leaves:	norm, abnorm
13.	leafspots-halo:	absent, yellow-halos, no-yellow-halos
14.	leafspots-marg:	w-s-marg, no-w-s-marg, dna
15.	leafspot-size:	lt-1/8, gt-1/8, dna
16.	leaf-shread:	absent, present
17.	leaf-malf:	absent, present
18.	leaf-mild:	absent, upper-surf, lower-surf
19.	stem:	norm, abnorm
20.	lodging:	yes, no
21.	stem-cankers:	absent, below-soil, above-soil, above-sec-nde
22.	canker-lesion:	dna, brown, dk-brown-blk, tan
23.	fruiting-bodies:	absent, present
24.	external decay:	absent, firm-and-dry, watery
25.	mycelium:	absent, present
26.	int-discolor:	none, brown, black
27.	sclerotia:	absent, present
28.	fruit-pods:	norm, diseased, few-present, dna
29.	fruit spots:	absent, colored, brown-w/blk-pecks, distort, dna
30.	seed:	norm, abnorm
31.	mold-growth:	absent, present
32.	seed-discolor:	absent, present
33.	seed-size:	norm, lt-norm
34.	shriveling:	absent, present
35.	roots:	norm, rotted, galls-cysts
36.	Classes:	diaporthe-stem-canker, charcoal-rot, rhizoctonia-root-rot, phytophthora-rot, brown-stem-rot, powdery-mildew, downy-mildew, brown-spot, bacterial-blight, bacterial-pustule, purple-seed-stain, anthracnose, phyllosticta-leaf-spot, alternarialeaf-spot, frog-eye-leaf-spot, diaporthe-pod-&-stem-blight, cyst-nematode, 2-4-d-injury, herbicide-injury.

The character ? is again used to mark missing values.

For the sake of clarity, we have omitted the ARFF file and are only showing a typical data record:

@DATA

october, normal, gt-norm, norm, yes, same-1st-yr, low-areas,
pot-severe, none, 90-100, abnorm, abnorm, absent, dna, dna,
absent,
absent, absent, abnorm, no, above-sec-nde, brown, present,
firm-and-dry, absent, none, absent, norm, dna, norm, absent,
absent, norm, absent, norm, diaporthes-stem-canker
....

A.3 Weather data

One of the classics in data mining is the data set for the problem concerning the weather conditions under which a game is played. With 14 data records, the data is certainly not mass data, but its small size makes it very well suited for illustrating data mining algorithms. There are two versions of this data.

Version 1

The first variant specifies the temperature and humidity numerically. The attributes are:

Outlook sunny, overcast, rainy

Temp in Fahrenheit

Humidity in %

Windy true, false

Play yes, no

Table A.1 Weather example with numerical data

Outlook	Temp (°F)	Humidity(%)	Windy	Play
sunny	85	85	false	no
sunny	80	90	true	no
overcast	83	78	false	yes
rainy	70	96	false	yes
rainy	68	80	false	yes
rainy	65	70	true	no
overcast	64	65	true	yes
sunny	72	95	false	no
sunny	69	70	false	yes
rainy	75	80	false	yes
sunny	75	70	true	yes
overcast	72	90	true	yes
overcast	81	75	false	yes
rainy	71	80	true	no

Version 2

The second version specifies the temperature and humidity ordinally. The attributes are:

Outlook sunny, overcast, rainy

Temp hot, mild, cool

Humidity high, normal

Windy true, false

Play yes, no

Table A.2 Weather example with nominal data

Day	Outlook	Temperature	Humidity	Windy	Play
1	sunny	hot	high	false	no
2	sunny	hot	high	true	no
3	overcast	hot	high	false	yes
4	rainy	mild	high	false	yes
5	rainy	cool	normal	false	yes
6	rainy	cool	normal	true	no
7	overcast	cool	normal	true	yes
8	sunny	mild	high	false	no
9	sunny	cool	normal	false	yes
10	rainy	mild	normal	false	yes
11	sunny	mild	normal	true	yes
12	overcast	mild	high	true	yes
13	overcast	hot	normal	false	yes
14	rainy	mild	high	true	no

A.4 Contact lens data

The data comes from J. Cendrowska⁰³. The goal is to predict whether a person needs contact lenses and, if so, which type of lenses (hard or soft) is preferable. The data is included in WEKA as sample data.

attribute 1 age of the patient: young, pre-presbyopic, presbyopic

attribute 2 spectacle prescription: myope, hypermetrope

attribute 3 astigmatic: no, yes

attribute 4 tear production rate: reduced, normal

attribute 5 class:

hard : the patient should be fitted with hard contact lenses

soft : the patient should be fitted with soft contact lenses

none : the patient should not be fitted with contact lenses

Here is the complete file in ARFF format:

```
@relation contact-lenses
```

```
@attribute age {young, pre-presbyopic, presbyopic}
```

```
@attribute spectacle-prescrip {myope, hypermetrope}
```

```
@attribute astigmatism      {no, yes}
@attribute tear-prod-rate    {reduced, normal}
@attribute contact-lenses    {soft, hard, none}
```

```
@data
```

```
% 24 instances
```

```
young, myope, no, reduced, none
young, myope, no, normal, soft
young, myope, yes, reduced, none
young, myope, yes, normal, hard
young, hypermetrope, no, reduced, none
young, hypermetrope, no, normal, soft
young, hypermetrope, yes, reduced, none
young, hypermetrope, yes, normal, hard
pre-presbyopic, myope, no, reduced, none
pre-presbyopic, myope, no, normal, soft
pre-presbyopic, myope, yes, reduced, none
pre-presbyopic, myope, yes, normal, hard
pre-presbyopic, hypermetrope, no, reduced, none
pre-presbyopic, hypermetrope, no, normal, soft
pre-presbyopic, hypermetrope, yes, reduced, none
pre-presbyopic, hypermetrope, yes, normal, none
presbyopic, myope, no, reduced, none
presbyopic, myope, no, normal, none
presbyopic, myope, yes, reduced, none
presbyopic, myope, yes, normal, hard
presbyopic, hypermetrope, no, reduced, none
presbyopic, hypermetrope, no, normal, soft
presbyopic, hypermetrope, yes, reduced, none
presbyopic, hypermetrope, yes, normal, none
Edition
```

© 2026 Walter de Gruyter GmbH, Berlin/Boston, Genthiner Straße 13, 10785
Berlin 2026 Walter de Gruyter GmbH, Berlin/Boston, Genthiner Straße 13,
10785 Berlin

Bibliography

Aal16 Wil van der Aalst. *Process Mining – Data Science in Action*. 2nd Edition. Berlin, Heidelberg: Springer, 2016. →

AIS93 Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. "Mining association rules between sets of items in large databases". In: *Proceedings of the ACM SIGMOD Conference on Management of Data, Washington D.C.* Ed. by P. Bunemann and S. Jajodia. New York: ACM Press, 1993, pp. 207–216. →

AS94 Rakesh Agrawal and Ramakrishnan Srikant. "Fast Algorithms for Mining Association Rules". In: *Proceedings 20th Int. Conf. Very Large Data Bases, VLDB*. Santiago, Chile: Morgan Kaufmann, 1994, pp. 487–499. →

Ber+20 Michael R. Berthold et al. *Guide to Intelligent Data Science - How to Intelligently Make Use of Real Data*. 2nd Edition. Berlin, Heidelberg: Springer, 2020.

Bez81 James C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. New York: Plenum Press, 1981. →

Bra20 Max Bramer. *Principles of Data Mining*. 4th Edition. London: Springer, 2020. →

Bre01 Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32. →

Bre96 Leo Breiman. "Bagging Predictors". In: *Machine Learning* 24.2 (1996), pp. 123–140. →

Chu14 Wesley W. Chu, ed. *Data Mining and Knowledge Discovery for Big Data*. Berlin, Heidelberg: Springer, 2014. →

Cio+10 Krzysztof J. Cios et al. *Data Mining: A Knowledge Discovery Approach*. New York: Springer, 2010.

CST00 Nello Christianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge: Cambridge University Press, 2000. →

CW25 François Chollet and Matthew Watson. *Deep Learning with Python*. 3rd Edition. Shelter Island, NY: Manning Publications, 2025. a, b

DLR77 Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm". In: *journal of the Royal Statistical Society Series B* 39 (1) (1977), pp. 1–38. →

Dun73 J. C. Dunn. "A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters". In: *journal of Cybernetics* Vol. 3 (1973), pp. 32–57. →

Est+96 Martin Ester et al. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1996, pp. 226–231.

FPSS96 Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. "From Data Mining to Knowledge Discovery: An Overview". In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Usama M. Fayyad et al. Menlo Park, Cambridge, London: MIT Press, 1996, pp. 1–34. a, b

Fre79 Gottlob Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens [Concept-Script: A Formal Language for Pure Thought Modeled on that of Arithmetic]*. In: Ignacio Angelelli (ed.): *Begriffsschrift und andere Aufsätze*, 6th Reprint of 2nd Edition (1964), Olms 2007. Halle a/S: Publisher Louis Nebert, 1879. →

Fri97 Bernd Fritzke. *Some Competitive Learning Methods*. Technical Report, Institut für Neuroinformatik, Ruhr University Bochum. 1997. URL: [→ https://www.demogng.de/](https://www.demogng.de/). →

Fri98 Bernd Fritzke. *Vektorbasierte neuronale Netze*. Aachen: Shaker, 1998. a, b, c

Fro21 Jörg Frochte. *Maschinelles Lernen - Grundlagen und Algorithmen in Python*. 3rd Edition. Munich: Hanser, 2021. →

FS07 Ronen Feldman and James Sanger. *The Text Mining Handbook - Advanced Approaches in Analyzing Unstructured Data*. Cambridge: Cambridge University Press, 2007. →

Han98 David J. Hand. "Consumer Credit and Statistics". In: *Statistics in Finance*. Ed. by David J. Hand and Saul D. Jacka. London: Hodder Arnold, 1998, pp. 69–81. →

Hip+01 Hajo Hippner et al., eds. *Handbuch Data Mining im Marketing: Knowledge Discovery in Marketing Databases*. Braunschweig, Wiesbaden: Vieweg, 2001.

HKP12 Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. 3rd Edition. Waltham: Morgan Kaufmann, 2012. →

Hon12 Kai Honsel. *Integrated Usage Mining - Eine Methode zur Analyse des Benutzerverhaltens im Web*. Berlin, Heidelberg, New York: Springer, 2012. →

Jo24 Taeho Jo. *Text Mining - Concepts, Implementation, and Big Data Challenge*. 2nd Edition. Cham: Springer Nature Switzerland, 2024. →

- KR90** Leonard Kaufman and Peter J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. New York: John Wiley and Sons, 1990. →
- Kru+91** Hilger Kruse et al. *Programmierung neuronaler Netze: Eine Turbo Pascal Toolbox*. Bonn, Munich u.a.: Addison-Wesley, 1991.
- LB01** Gordon S. Linoff and Michael J. A. Berry. *Mining the Web*. New York: John Wiley & Sons, 2001. →
- LC23** Uwe Lämmel and Jürgen Cleve. *Künstliche Intelligenz*. 6th Edition. Munich: Hanser, 2023. a, b, c, d, e, f
- Liu11** Bing Liu. *Web Data Mining*. 2nd Edition. Berlin, Heidelberg: Springer, 2011. →
- Mac67** James B. MacQueen. "Some Methods for Classification and Analysis of Multivariate Observations". In: *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*. Ed. by L. Le Cam and J. Neyman. Vol. 1. University of California Press, 1967, pp. 281–297. →
- MN73** James Moore and Allen Newell. "How can Merlin understand?" In: *Knowledge and Cognition*. Ed. by Lee W. Gregg. Lawrence Earlbaum Hillsdale N.J., 1973, pp. 201–252. →
- NH94** Raymond T. Ng and Jiawei Han. "Efficient and Effective Clustering Methods for Spatial Data Mining". In: *Proceedings 20th Int. Conf. Very Large Data Bases, VLDB*. Santiago, Chile: Morgan Kaufmann, 1994, pp. 144–155. →
- NK25** Klaus North and Gita Kumta. *Knowledge Management: Value Creation Through Organizational Learning*. 3rd Edition. Springer Texts in Business and Economics. Cham: Springer Nature Switzerland, Imprint: Springer, 2025. DOI: 10.1007/978-3-031-87650-9. →
- PA23** Alejandro Peña-Ayala. *Educational Data Science: Essentials, Approaches, and Tendencies*. Singapore: Springer Nature, 2023. →
- Pet05** Helge Petersohn. *Data Mining*. Munich: Oldenbourg, 2005. a, b
- PF91** Gregory Piatetski and William Frawley. *Knowledge Discovery in Databases*. Cambridge, MA, USA: MIT Press, 1991.
- PM23** David Poole and Alan Mackworth. *Artificial Intelligence*. 3rd Edition. Cambridge, USA: Cambridge University Press, 2023. URL: [→ https://artint.info/3e/html/ArtInt3e.html](https://artint.info/3e/html/ArtInt3e.html). →
- PS91** Gregory Piatetsky-Shapiro. "Discovery, analysis and presentation of strong rules". In: *Knowledge Discovery in Databases*. Ed. by Gregory Piatetsky-Shapiro and William J. Frawley. AAAI Press, 1991, pp. 229–248. →

Qui86 Ross Quinlan. "Induction of Decision Trees". In: *Machine Learning* 1 (1986), pp. 81–106. →

Qui93 Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993. →

RK89 Helge Ritter and Teuvo Kohonen. *Self-organizing semantic maps*. Otaniemi, 1989. →

Rom+10 Cristobal Romero et al. *Handbook of Educational Data Mining*. Boca Raton: CRC Press, 2010.

Rou87 Peter Rousseeuw. "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis". In: *J. Comput. Appl. Math.* 20.1 (1987), pp. 53–65. →

Run25 Thomas A. Runkler. *Data Analytics - Models and Algorithms for Intelligent Data Analysis*. 4th Edition. Wiesbaden: Springer, 2025.

SA95 Ramakrishnan Srikant and Rakesh Agrawal. "Mining Generalized Association Rules". In: *Proceedings of the 21th International Conference on Very Large Data Bases*. VLDB '95. San Francisco: Morgan Kaufmann, 1995, pp. 407–419. →

SA96 Ramakrishnan Srikant and Rakesh Agrawal. "Mining Quantitative Association Rules in Large Relational Tables". In: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4–6, 1996*. Ed. by H. V. Jagadish and Inderpal Singh Mumick. ACM Press, 1996, pp. 1–12. →

Sch90 Robert E. Schapire. "The strength of weak learnability". In: *Machine Learning* 5.2 (1990), pp. 197–227. URL: [→ https://rob.schapire.net/papers/strengthofweak.pdf](https://rob.schapire.net/papers/strengthofweak.pdf) (visited on 03/24/2026). →

Sha26 Chirag Shah. *A Hands-On Introduction to Data Science*. 2nd Edition. Cambridge: Cambridge University Press, 2026. →

Sha48 Claude Elwood Shannon. "A mathematical theory of communication". In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. →

SMA19 N. N. R. Ranga Suri, Narasimhaury M, and G. Athithan. *Outlier Detection: Techniques and Applications - A Data Mining Perspective*. Berlin, Heidelberg: Springer, 2019.

Soa+08 Carlos Soares et al., eds. *Applications of Data Mining in E-Business and Finance*. Amsterdam, Berlin, Oxford, Tokyo, Washington DC: IOS Press, 2008.

SS25 Joachim Steinwendner and Roland Schwaiger. *Neuronale Netze programmieren mit Python*. 3rd Edition. Bonn: Rheinwerk, 2025. →

War63 Joe H. Ward. "Hierarchical Grouping to Optimize an Objective Function". In: *Journal of the American Statistical Association* 58 (1963), pp. 236–244. →

Wit+25 Ian H. Witten et al. *Data Mining - Practical Machine Learning Tools and Techniques*. 5th Edition. Morgan Kaufmann, 2025.

Yau24 Nathan Yau. *Visualize This: The FlowingData Guide to Design, Visualization, and Statistics*. 2nd Edition. Indianapolis: Wiley, 2024. →

Zel97 Andreas Zell. *Simulation Neuronaler Netze*. Munich: Oldenbourg, 1997. a, b, c

Index Rerum

activation function [1](#), [2](#), [3](#)
adaptive resonance theory [1](#)
agglomerative clustering [1](#), [2](#), [3](#)
– algorithm [1](#)
aggregation [1](#), [2](#)
Anaconda [1](#)
AND operator [1](#)
application classes [1](#)
Apriori algorithm [1](#), [2](#)
– generating candidates [1](#)
– generating rules [1](#)
– join [1](#)
– pruning [1](#)
area under curve [1](#)
ARFF [1](#)
ART *see* adaptive resonance theory
artificial intelligence [1](#)
artificial neural network *see* neural network
artificial neuron [1](#), [2](#)
association analysis [1](#), [2](#)
– Apriori algorithm [1](#)
– frequent pattern growth [1](#)
association rule [1](#), [2](#)
– evaluation [1](#)
– fuzzy [1](#)
– hierarchical [1](#), [2](#)

- quantitative [1](#), [2](#)
- temporal [1](#), [2](#)
- association rule mining [1](#)
- AUC [1](#)
- average linkage [1](#)

- backpropagation network [1](#)
- backpropagation of error [1](#)
- algorithm [1](#)
- QuickProp [1](#)
- resilient propagation [1](#)
- with momentum [1](#)
- bagging [1](#)
- bar chart [1](#)
- Bayesian formula [1](#)
- best matching unit see winner neuron
- binary coding [1](#), [2](#)
- binning [1](#), [2](#), [3](#)
- boosting [1](#)
- bootstrapping [1](#)
- box plot [1](#)
- bubble chart [1](#)
- business intelligence [1](#)

- C 4.5 [1](#)
- centroid [1](#), [2](#), [3](#)
- Chebyshev distance [1](#)
- CLARA [1](#)
- CLARANS [1](#)
- classification [1](#), [2](#)
- evaluation [1](#)
- classification rule [1](#), [2](#), [3](#)
- cluster [1](#), [2](#), [3](#)

cluster analysis see clustering clustering [1](#), [2](#)

– adaptive resonance theory [1](#)

– agglomerative [1](#), [2](#)

– density-based [1](#), [2](#)

– divisive [1](#)

– evaluation [1](#)

– expectation maximization [1](#)

– fuzzy c-means [1](#)

– hierarchical [1](#)

– neural gas [1](#)

– self-organizing map [1](#)

– with neural networks [1](#)

competitive learning [1](#), [2](#)

complete linkage [1](#)

completeness [1](#)

comprehensibility [1](#), [2](#)

confidence [1](#), [2](#), [3](#), [4](#), [5](#)

confusion matrix [1](#)

conjunction [1](#)

convolutional neural network [1](#)

cost matrix [1](#)

CRISP data mining model [1](#)

cross-validation [1](#)

data [1](#)

– semi-structured [1](#)

– structured [1](#)

– unstructured [1](#)

data cleaning [1](#)

data compression [1](#)

data integration [1](#)

data mining [1](#), [2](#), [3](#)

– fundamentals [1](#)

- process [1](#), [2](#)
- subfields [1](#)
- data mining tools [1](#)
- data preparation [1](#), [2](#), [3](#)
- data preprocessing [1](#), [2](#), [3](#), [4](#)
- data reduction [1](#)
- numerical [1](#)
- data science [1](#)
- data selection [1](#), [2](#), [3](#)
- data smoothing [1](#)
- data transformation [1](#), [2](#), [3](#), [4](#)
- data type [1](#)
- metric [1](#)
- nominal [1](#)
- ordinal [1](#)
- data understanding [1](#)
- data warehouse [1](#), [2](#)
- database [1](#)
- DBScan [1](#)
- decision table [1](#)
- decision tree [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#)
- C 4.5 [1](#)
- ID 3 [1](#)
- random forest [1](#), [2](#)
- selection of an attribute [1](#)
- decision tree learning see decision tree deep learning [1](#), [2](#)
- DemoGNG [1](#)
- dendrogram [1](#)
- density-based clustering [1](#), [2](#), [3](#)
- descriptive analytics [1](#)
- diagnostic analytics [1](#)
- dimension reduction [1](#)
- discretization [1](#)

disjunction [1](#)
distance [1](#), [2](#)
- Chebyshev [1](#)
- Euclidian [1](#)
- weighted [1](#)
- Hamming [1](#)
- Manhattan [1](#)
- maximum [1](#)
- Minkowski [1](#)
distance function [1](#)
distance measure *see* distance
divisive clustering [1](#)

eager learner [1](#)
ensemble learning [1](#), [2](#)
- bagging [1](#)
- boosting [1](#)
entropy [1](#), [2](#)
equivalence [1](#)
error costs [1](#)
error rate [1](#), [2](#)
Euclidian distance [1](#)
- weighted [1](#)
evaluation [1](#), [2](#), [3](#)
- association rule [1](#)
- classification [1236](#)
 - - F-measure [2](#)
 - - precision [2](#)
 - - recall [2](#)
- clustering [1](#), [2](#)
example set [1](#)
existence quantification [1](#)
expectation maximization [1](#), [2](#)

F-measure [1](#)
Fayyad's model [1](#)
feedforward neural network [1](#), [2](#), [3](#), [4](#), [5](#), [6](#)
frequent item set [1](#)
frequent pattern growth [1](#), [2](#)
fuzzy association rule [1](#)
fuzzy logic [1](#)
fuzzy c-means [1](#), [2](#)
gain function [1](#)
Gaussian distribution [1](#)
Gini index [1](#)
goal function [1](#)

Hamming distance [1](#)
hierarchical association rule [1](#), [2](#)
hierarchical clustering [1](#)
histogram [1](#)
holdout [1](#)
hyperbolic tangent [1](#), [2](#)

ID 3 [1](#)
implication [1](#)
implicit knowledge [1](#)
imputation [1](#)
inconsistent data [1](#)
information [1](#)
instance set [1](#), [2](#)
instance-based learning [1](#)
instance-based representation [1](#)
interdisciplinarity [1](#)
– business intelligence [1](#)
– data warehouse [1](#)
– database [1](#)

- knowledge-based systems [1](#)
- machine learning [1](#)
- statistics [1](#)
- visualization [1](#)

interestingness [1](#)

interestingness measure [1](#), [2](#)

interpretation [1](#), [2](#)

item [1](#)

item set [1](#)

JAVANNS [1](#), [2](#), [3](#)

Jupyter Notebook [1](#)

KDD [1](#)

k-means [1](#), [2](#), [3](#)

- algorithm [1](#), [2](#)

k-medoid [1](#), [2](#)

- algorithm [1](#)
- CLARANS [1](#)
- PAM [1](#)

k-nearest neighbor [1](#), [2](#), [3](#), [4](#)

- algorithm [1](#)

KNIME 18, [1](#)

knowledge [1](#)

knowledge discovery in databases [1](#)

knowledge representation [1](#)

knowledge-based systems [1](#)

Kohonen net *see* self-organizing map

lazy learner [1](#), [2](#)

learning

- competitive [1](#)
- supervised [1](#), [2](#), [3](#), [4](#), [5](#)

- unsupervised [1](#), [2](#), [3](#), [4](#), [5](#)
- leave-one-out [1](#)
- lift [1](#)
- likelihood [1](#), [2](#)
- line chart [1](#)
- linear regression [1](#), [2](#), [3](#)
- linear separability [1](#)
- logic [1](#)
- logical formula [1](#)
- logical rule [1](#)
- logistic function [1](#)
- machine learning [1](#)
- Manhattan distance [1](#)
- market basket analysis [1](#)
- maximum distance [1](#)
- medoid [1](#), [2](#), [3](#)
- metric data type [1](#)
- minimum description length [1](#)
- Minkowski distance [1](#)
- missing data see missing values missing values [1](#)
- MAR [1](#)
- MCAR [1](#)
- MNAR [1](#)
- model [1](#)
- multilayer perceptron [1](#), [2](#)

- naive Bayes [1](#), [2](#), [3](#)
- negation [1](#)
- neural gas [1](#)
- neural network [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#)
- adaptive resonance theory [1](#)
- convolutional neural network [1](#)
- neural gas [1](#)

- self-organizing map [1](#)
- neuron [1](#), [2](#)
- noisy data [1](#)
- nominal data type [1](#)
- normalization [1](#)
- novelty [1](#), [2](#)
- numerical data reduction [1](#)
- numerical prediction [1](#)

- objective function [1](#), [2](#)
- Occam's razor [1](#)
- one-hot encoding [1](#)
- OR operator [1](#)
- ordinal data type [1](#)
- outlier [1](#), [2](#), [3](#)
- output function [1](#), [2](#)
- overfitting [1](#), [2](#), [3](#)
- oversampling [1](#)

- PAM [1](#)
- perceptron [1](#)
- pie chart [1](#)
- plasticity-stability dilemma [1](#)
- PR diagram [1](#)
- precision [1](#)
- predictive analytics [1](#)
- prescriptive analytics [1](#)
- principal component analysis [1](#), [2](#)
- probability [1](#), [2](#), [3](#)
- propagation function [1](#)
- pruning [1](#), [2](#)
- p -s function [1](#)
- Python [1](#), [2](#)

quantitative association rule [1](#), [2](#)

random forest [1](#), [2](#)

recall [1](#)

receiver operating characteristic [1](#)

receiver operating curve [1](#)

regression [1](#)

regression function [1](#)

regression tree [1](#)

reinforcement learning [1](#)

ReLU function [1](#)

ROC [1](#)

rule [1](#), [2-3](#), [4](#)

sample [1](#)

scaling [1](#), [2](#)

scatter matrix [1](#)

scatter plot [1](#)

self-organizing map [1](#)

– learning [1](#)

SEMMA model [1](#)

sensitivity [1](#)

Shepard's method [1](#)

similarity [1](#)

similarity measure [1](#)

– cosine [1](#)

single linkage [1](#), [2](#)

SMOTE [1](#)

SOM see self-organizing map split info [1](#)

standardization [1](#)

statistics [1](#)

stratification [1](#)

subfields of data mining [1](#)

success rate [1](#)
supervised learning [1](#), [2](#), [3](#), [4](#)
support [1](#), [2](#), [3](#), [4](#), [5](#)
support vector [1](#)
support vector machine [1](#), [2](#)

taxonomy [1](#)
TDIDT [1](#), [2](#)
temporal association rule [1](#), [2](#)
test set [1](#), [2](#)
text mining [1](#), [2](#)
threshold [1](#)
threshold function [1](#)
threshold value [1](#), [2](#)
training set [1](#)

undersampling [1](#)
universal quantification [1](#)
unsupervised learning [1](#), [2](#), [3](#)
usefulness [1](#), [2](#)

validation set [1](#)
validity [1](#), [2](#)
visualization [1](#), [2](#)
– bar chart [1](#)
– box plot [1](#)
– bubble chart [1](#)
– histogram [1](#)
– line chart [1](#)
– pie chart [1](#)
– scatter matrix [1](#)
– scatter plot [1](#)

web mining [1](#), [2](#)

- web content mining [1](#)
- web log mining [1](#)
- web structure mining [1](#)
- web usage mining [1](#)
- weighted Euclidian distance [1](#)
- WEKA [1](#)
- winner neuron [1](#)

Notes

- 1 The DMC is a student competition organized jointly by Chemnitz University of Technology and the company *prudsys AG*, Chemnitz, Germany. The task is to analyze real company data.
- 2 Now: GK Artificial Intelligence AG, part of the GK Software Group since 2017
- 1 At [→https://www.kdnuggets.com/news/2000/n13/23i.html](https://www.kdnuggets.com/news/2000/n13/23i.html), it is reported that this example was invented by Tom Blishok (a retail consultant) around 1992. Apparently, there was never any real analysis.
- 2 Download at [→https://www.knime.com](https://www.knime.com)
- 3 We provide JAVANNS on the book's website at [→https://www.fww.hs-wismar.de/datamining/](https://www.fww.hs-wismar.de/datamining/).
- 4 [→https://www.anaconda.com](https://www.anaconda.com), last accessed on February 13, 2026.
- 1 Arity refers to the number of arguments of a function or predicate. n -ary thus means n arguments.
- 1 Discrete function: a function with a finite range of values
- 1 Representation based on [Ber+20, p. 183].)
- 2 SoKo Wismar (Self-Organizing Kohonen map) is available on the book's webpage [→https://www.fww.hs-wismar.de/datamining/](https://www.fww.hs-wismar.de/datamining/).

- 3** See [→https://www.demogng.de/](https://www.demogng.de/), last accessed on March 25, 2026.
- 1** We use set notation here to clarify that the frequent item sets from *MP* can freely draw from the items in *P*. We use the abbreviation *af* for $\{a, f\}$.
- 1** PCA was developed by Karl Pearson (1901).
- 1** Division by 0 is, of course, undefined and is set to ∞ by us.
- 2** Created with gnuplot [→http://gnuplot.info/](http://gnuplot.info/)
- 1** The data is available for download on the book's website [→https://www.fww.hs-wismar.de/datamining/](https://www.fww.hs-wismar.de/datamining/).
- 1** Edgar Anderson (1935). "The irises of the Gaspé Peninsula". In: Bulletin of the American Iris Society 59: 2-5
- 2** R. S. Michalski and R. L. Chilausky. Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis, International Journal of Policy Analysis and Information Systems, Vol. 4, No. 2, 1980.
- 03** Cendrowska, J.: PRISM: An algorithm for inducing modular rules, International Journal of Man- Machine Studies, 1987, 27, 349–370.